# UML + Casi di studio

UNIVERSITÀ DELLA CALABRIA

DIMES - Dipartimento di INGEGNERIA INFORMATICA
MODELLISTICA, ELETTRONICA E SISTEMISTICA

Ing. Ludovica Sacco
DIMES – UNICAL - 87036 Rende(CS) - Italy
Email: l.sacco@dimes.unical.it

# UML

# Unified Modeling Language

UML è un linguaggio grafico che supporta la progettazione e la descrizione di sistemi software:

- Si presta particolarmente alla descrizione di sistemi software costruiti secondo il paradigma orientato agli oggetti
- Permette la descrizione del sistema software ad un alto livello di astrazione

È indipendente dal linguaggio di programmazione e dalla tecnologia usate.

# Principali diagrammi UML

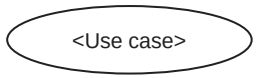
- Structural models:
  - Class diagrams
  - Deployment diagrams
  - Package diagrams

- Behavioural models:
  - Use case diagrams
  - Activity diagrams
  - Sequence diagrams
  - State diagrams

# Use Case Diagram: elementi

Sono diagrammi che descrivono le *funzioni* o *servizi* offerti da un sistema, evidenziando le interazioni tra gli *attori* del sistema.


- Actor → identificato da un'icona che rappresenta un uomo stilizzato. Rappresenta un *ruolo* coperto da un insieme di entità interagenti col sistema.




- Use Case → rappresentato con un'ellisse contenente il nome del caso d'uso. Rappresenta una *funzione* o *servizio* offerto dal sistema a uno o più attori.

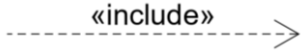
- Sistema → contiene actor e use case, in genere rappresentato con un rettangolo vuoto, ma spesso omesso. Quando rappresentato permette di visualizzare gli elementi che interagiscono al suo interno e le interazioni con gli elementi posti all'esterno.

# Use Case Diagram: relazioni

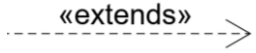
- **Associazioni** → relazione che congiunge gli attori con i casi d'uso a cui essi partecipano. Un attore può essere associato a un qualsiasi numero di casi d'uso e viceversa.

- **Generalizzazioni** → sia per gli attori che per i casi d'uso possono esistere delle relazioni di generalizzazione (Principio di sostituzione di Liskov). Un «sottoattore» deve essere in grado di partecipare a tutti i casi d'uso a cui partecipa il «superattore», eventualmente può partecipare a casi d'uso aggiuntivi o partecipare in modo diverso a qualche use case ''ereditato''. Un «sotto-caso d'uso» deve fornire lo stesso servizio generale del «super-caso d'uso», eventualmente producendo valore aggiuntivo o fornendolo a qualche tipologia di attore aggiuntiva o seguendo un procedimento parzialmente diverso per ottenere il risultato e così via.

# Use Case Diagram: relazioni




- **Inclusione** → rappresentata da una linea tratteggiata con un'indicazione sulla freccia «includes», indica che la funzione rappresentata da uno dei due use case (quello alla base della freccia) include completamente la funzione rappresentata dall'altro (quello alla punta).



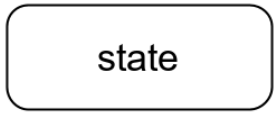

- **Estensione** → La relazione di estensione fra use case, rappresentata da una linea tratteggiata con indicazione dello stereotipo «extends», indica che la funzione rappresentata dallo use case "estendente" (alla base della freccia) può essere impiegata nel contesto della funzione "estesa" (lo use case alla punta), ovvero ne rappresenta una sorta di arricchimento.

# Activity Diagram: elementi e flusso

L’activiy diagram modella un processo. Organizza più entità in un insieme di azioni secondo un determinato flusso. Può essere sequenziale o concorrente.

**Initial node** → Il nodo iniziale rappresenta il punto iniziale del diagramma di attività.

**Activity final node** → Il nodo finale dell'attività rappresenta il punto di terminazione dell'attività.



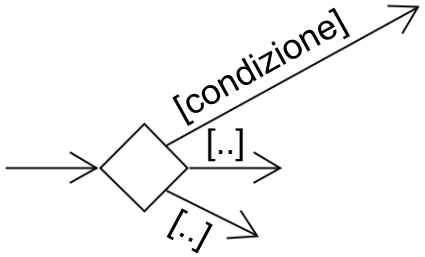

**Action state** → è un tipo di nodo di attività che rappresenta una singola azione o comportamento dell'attività da modellare.


**Object** → sono oggetti particolarmente importanti usati come input e output di azioni.

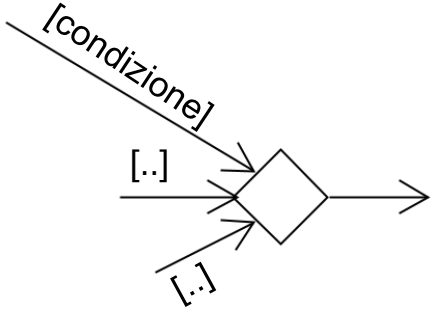
# Activity Diagram: elementi e flusso



**Activity edge** → Mostrano la transizione da un'attività a un'altra

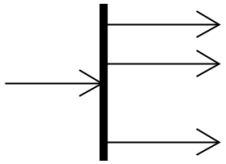



**Decision** → Ha un flusso in entrata e più uscite. I flussi in uscita hanno ciascuno una condizione che deve essere soddisfatta per attraversare il flusso

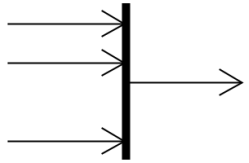



**Merge** → Ha diversi flussi in entrata e uno in uscita. L'unione indica che più flussi sono indirizzati in un unico punto.

# Activity Diagram: elementi e flusso



**Fork** → ha un flusso in entrata e più uscite. Indica che si stanno verificando più processi in parallelo

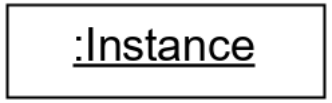


**Join** → ha diversi flussi in entrata e uno in uscita. Indica che più flussi paralleli si stanno unendo in un singolo punto. Tutti i flussi che entrano nel join devono essere completati prima che possa iniziare l'attività successiva.

**Final flow** → Quando raggiunti dall'arco di un'attività, causano la terminazione solo del flusso che li ha toccati

# Sequence Diagram: elementi

Appartiene alla categoria dei diagrammi di **interazione**, che descrivono come degli oggetti collaborano per implementare un certo comportamento.

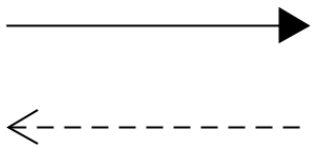



**Istanza** → Oggetto, istanza di attori o partecipante alle interazioni



**Linee di vita** → posta al di sotto del rettangolo dell'istanza, su di essa hanno luogo gli eventi

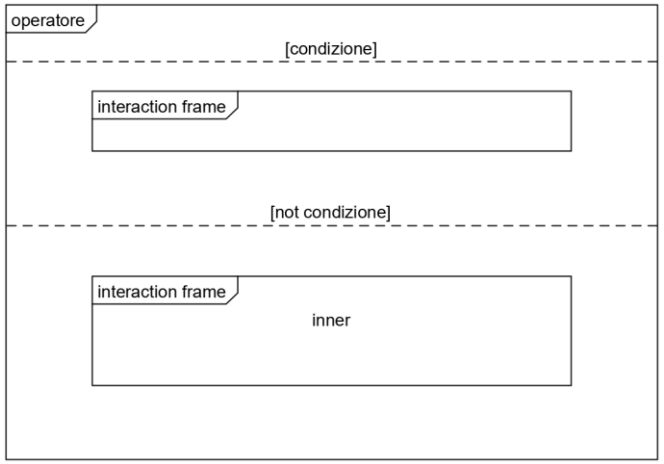
**Barra di attivazione** → Indica quando il partecipante è attivo nell'interazione



**Messaggi** → Rappresentano comunicazioni tra linee di vita: segnali, chiamate di operazioni, creazione e distruzione di oggetti.

# Sequence Diagram: frame di interazione

Le sequenze degli eventi in un caso d'uso contengono parole chiave come if, then, else, for e while.
Si ricorre ai frame di interazione (o frammenti combinati).




Un frame di interazione è una sotto area di un diagramma di sequenza che racchiude una parte dell'interazione e le modalità della sua esecuzione.

Gli elementi di un frame di interazione sono:

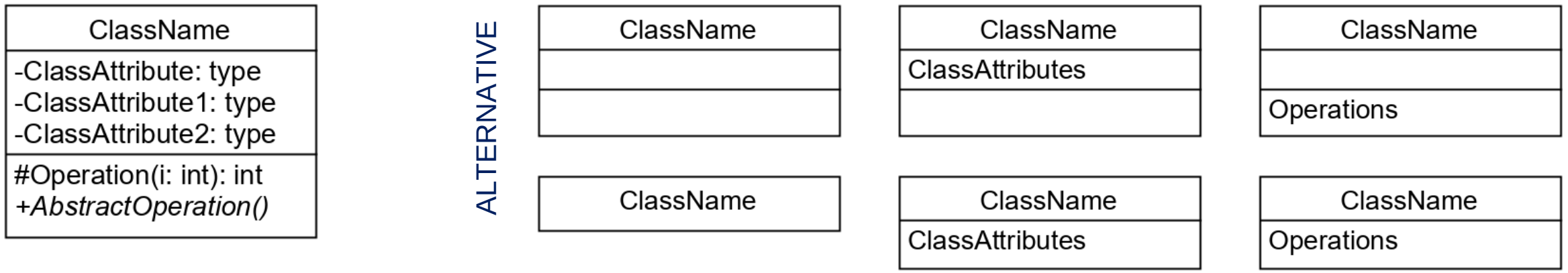
- un *operatore*, che specifica come il frammento viene eseguito (es. opt, alt, loop,etc.)
- uno o più *operandi*, sotto aree del diagramma di sequenza che possono essere eseguite
- *condizioni di guardia opzionali*, espressioni che determinano quali operandi sono eseguiti.

# Sequence Diagram: frame di interazione (operatori)

| Operatore | Significato |
|---|---|
| alt | Frammenti multipli in alternativa; si esegue solo quello per cui la condizione è soddisfatta. |
| opt | Opzionale; il frammento è eseguito solo se la condizione è verificata. |
| par | Parallelo; ogni frammento è eseguito in parallelo. |
| loop | Ciclo; il frammento può essere eseguito più volte, la guardia indica la condizione di iterazione. |
| region | Regione critica; il frammento può essere eseguito da un solo thread alla volta |
| neg | Negativo; il frammento mostra un'interazione non valida. |
| ref | Riferimento; si riferisce ad un'interazione mostrata in un altro diagramma. |
| sd | Sequence Diagram; usato per racchiudere un intero diagramma di sequenza. |

# Class Diagram: classi

Il class diagram rappresenta uno schema concettuale e fornisce una vista strutturale in termini di classi e relazioni.



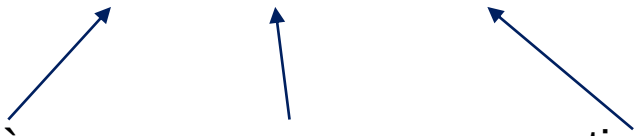

**Class** → è necessario specificare il nome di una classe, ma non è necessario riempire o mostrare ogni sezione (ALTERNATIVE)

# Class Diagram: classi (attribute e metodi)

**Attributi**: solo il nome è necessario

```
Visibilità nome : tipo [molteplicità] = default
```

es. # cognome : String = valoreIniziale




**Metodi**: solo il nome è necessario e non ci possono essere due operazioni con la stessa signature

```
Visibilità nome (lista-parametri) : tipo-ritorno
```

es. + aumentaStipendio (aumento : double) : double

# Class Diagram: indicatori di visibilità

**Indicatori di visibilità**

+ public → ogni elemento che può accedere alla classe può anche accedere a ogni suo membro con visibilità pubblica

- private → solo le operazioni della classe possono accedere ai membri con visibilità privata

# protected → solo le operazioni appartenenti alla classe o ai suoi discendenti possono accedere ai membri con visibilità protetta

~ package → ogni elemento nello stesso package della classe (o suo sottopackage annidato) può accedere ai membri della classe con visibilità package

# Class Diagram: molteplicità

**Molteplicità**

n…m → da n a m (n e m sono rispettivamente il limite inferiore e superiore

1 → uno (e uno solo), abbreviazione per 1…1

0…1 → zero oppure uno

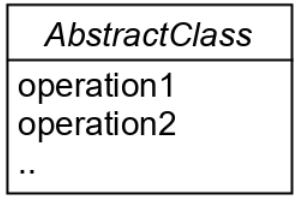
0…* → da zero a qualsiasi intero positivo

* → abbreviazione per 0…*

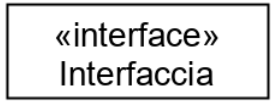
1…* → da 1 a qualsiasi intero positivo

Quando la molteplicità è omessa si assume sia 1.

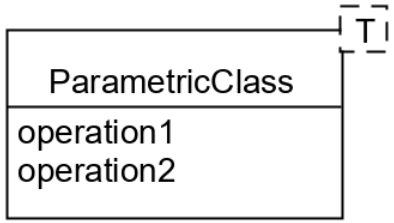
# Class Diagram: altri elementi



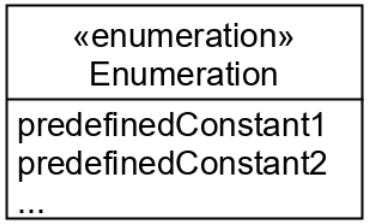

**Classi astratte** → il nome della classe è corsivo




**Interfacce** → presentano la parola chiave interface posta tra <<...>>



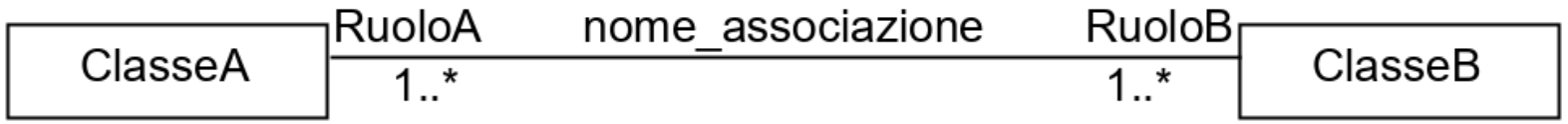

**Classi parametriche** → presentano lo stessa rappresentazione di una normale classe, con l'aggiunta di un riquadro tratteggiato contenente il parametro




**Enumerazioni** → presentano la parola chiave enumeration posta tra <<…>>

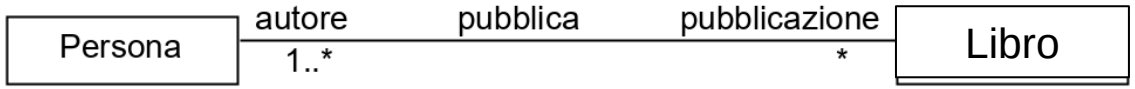
# Class Diagram: associazioni

**Associazioni** (collegamento tra classi)

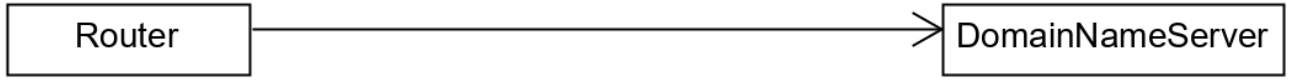



Esempi:

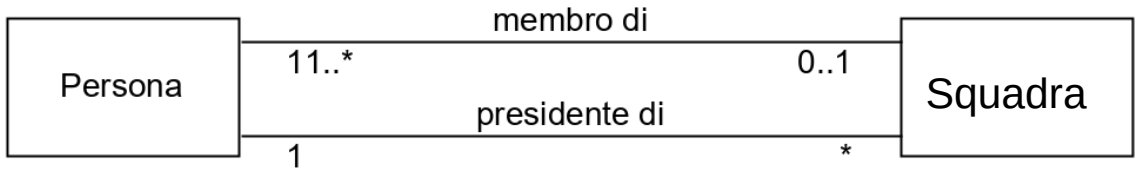
Associazione bidirezionale



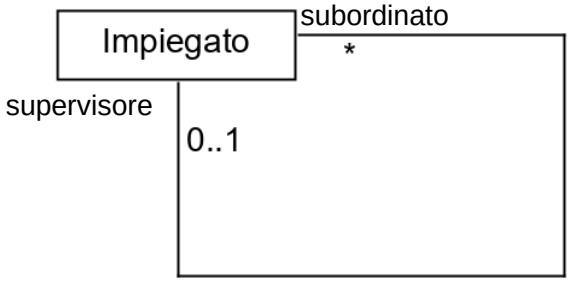

Associazione unidirezionale




Associazione doppia



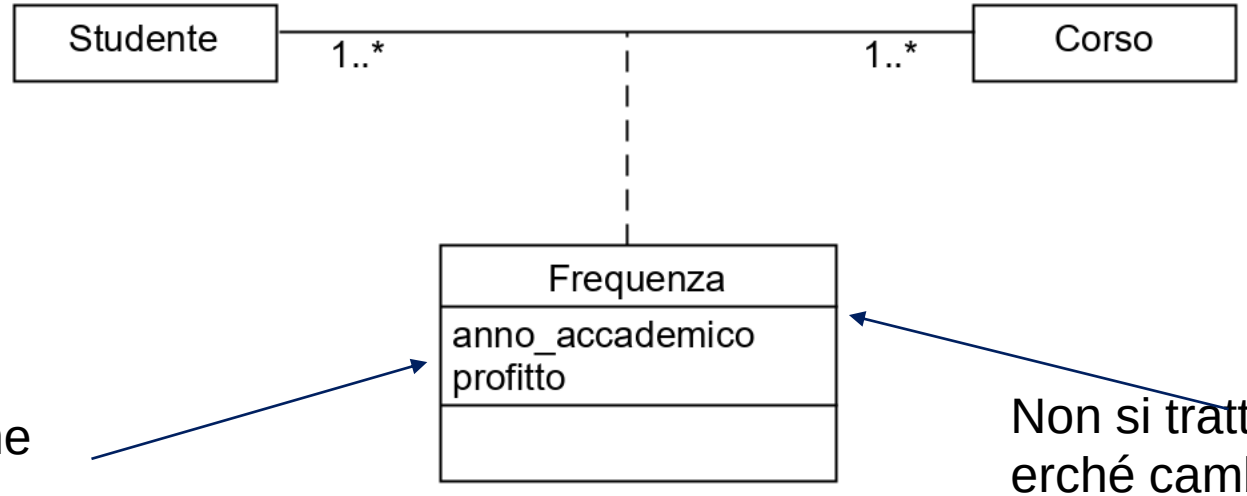

Associazione riflessiva



# Class Diagram: classi associative

La *classe associative* è una classe derivata da una associazione, che permette di aggiungere attributi a una associazione piuttosto che alle classi coinvolte.

Esse sottintendono un vincolo aggiuntivo, ossia il fatto che ci può essere solo un'istanza della classe di associazione fra ogni coppia di oggetti associati.




Con questa notazione indichiamo che l'associazione possiede alcuni attributi.

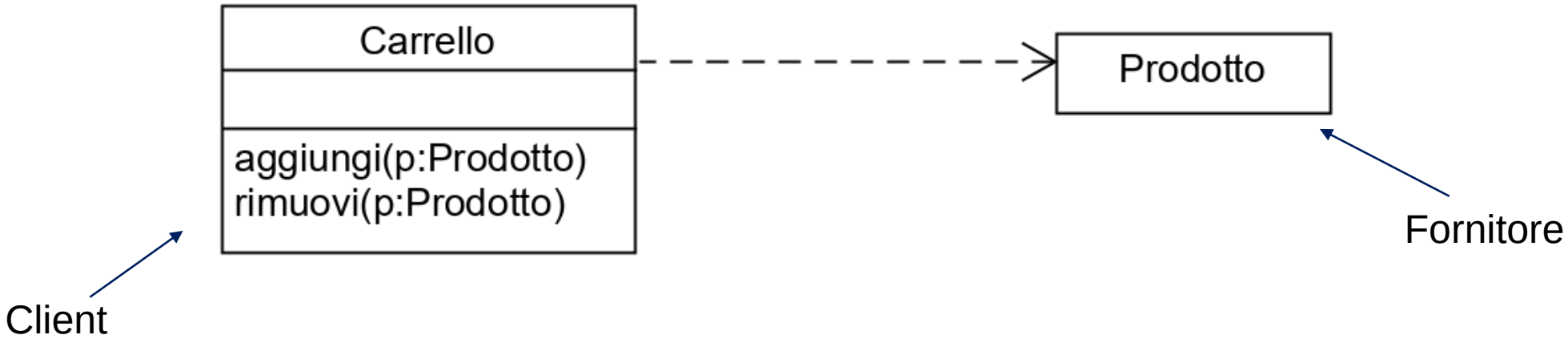
Non si tratta di attributi dello studente perché cambiano da corso a corso. Né sono attributi del corso (ad es. ogni corso è frequentato da studenti diversi in anni diversi)

# Class Diagram: dipendenze

**Dipendenze**
Una dipendenza è unidirezionale e non è una relazione transitiva.

Tra due elementi di un diagramma esiste una relazione di dipendenza se una modifica alla definizione di uno (fornitore o supplier) può comportare un cambiamento all'altro (il client)

La tipologia di dipendenza può essere specificata con uno stereotipo (<<call>>,<<create>>, etc.)

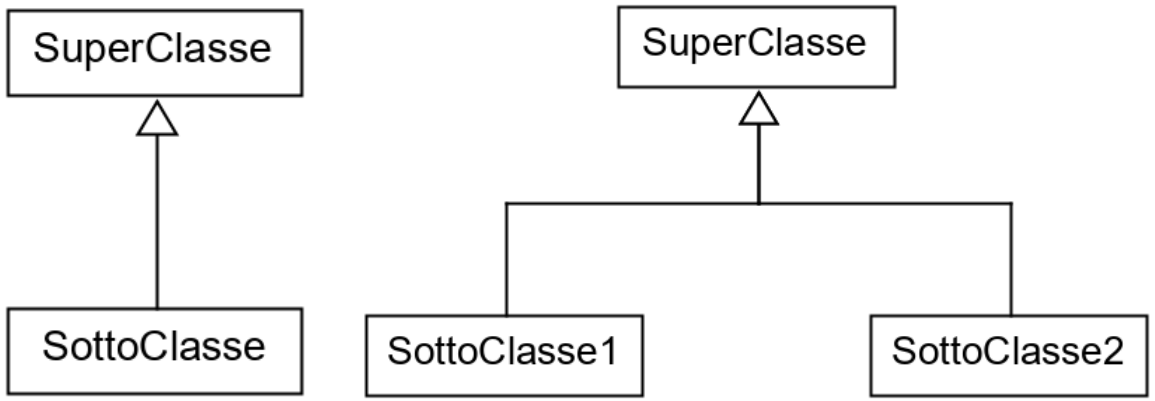


# Class Diagram: dipendenze (stereotipi)

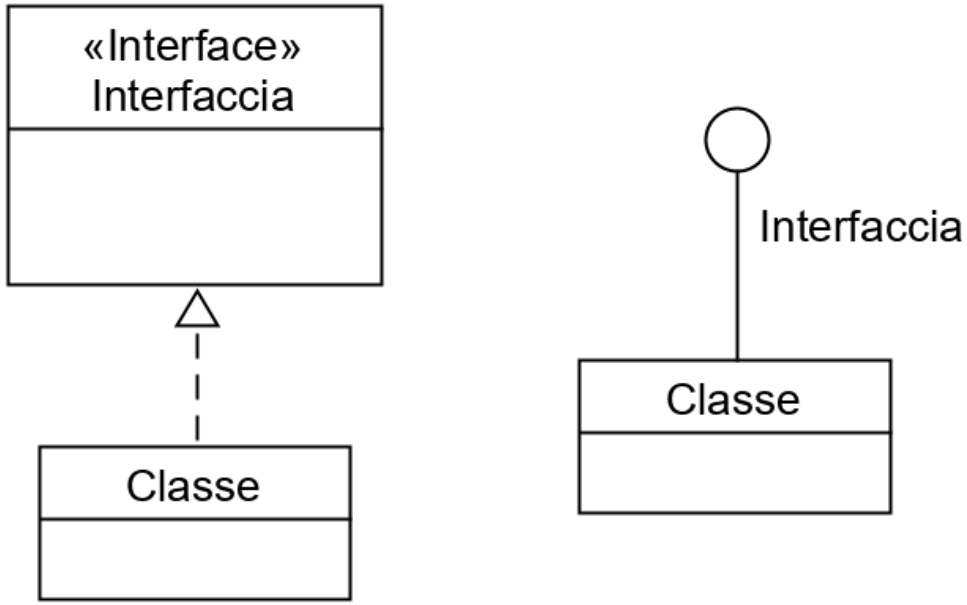
| Parola chiave | Significato |
|---|---|
| <<call>> | La sorgente invoca un'operazione della classe destinazione |
| <<create>> | La sorgente crea istanze della classe destinazione |
| <<derive>> | La sorgente è derivate dalla classe destinazione |
| <<instantiate>> | La sorgente è un'istanza della classe destinazione. (In questo caso se la sorgente è una classe, ed è istanza della destinazione, quest'ultima deve essere una meta-classe) |
| <<permit>> | La classe destinazione permette alla sorgente di accedere ai suoi campi privati |
| <<realize>> | La sorgente è un'implementazione di una specifica o di un'interfaccia definita dalla destinazione (si veda in seguito) |
| <<refine>> | Il raffinamento indica una relazione tra livelli semantici differenti; la classe sorgente ad esempio potrebbe essere una classe di progettazione, la destinazione una classe di analisi |
| <<substitute>> | La sorgente è sostituibile alla destinazione (si veda in seguito) |
| <<trace>> | Usata per tenere traccia di cose come i requisiti o di come i cambiamenti a una parte di modello si collegano ad altre sue parti |
| <<use>> | La sorgente richiede la destinazione per la sua implementazione |

# Class Diagram: generalizzazioni e realizzazioni

**Generalizzazioni** (extends)




**Realizzazioni** (implements)

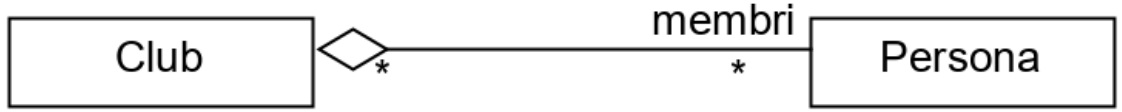


# Class Diagram: aggregazione e composizione

Servono a modellare associazioni tra un oggetto composto (il “tutto”) e gli oggetti che lo compongono (le “parti”)

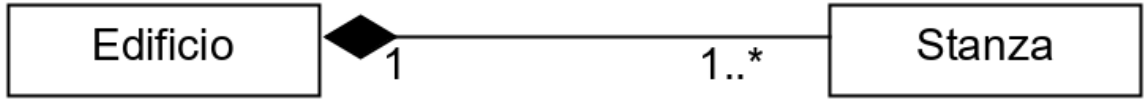
**Aggregazione**

- Le parti possono esistere indipendentemente dal tutto
- Una parte potrebbe appartenere a più oggetti composti
- Le parti potrebbero non essere essenziali per il tutto



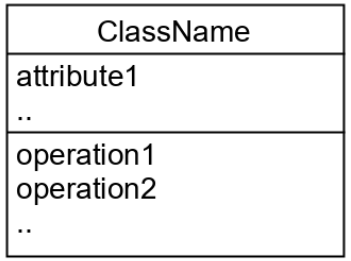

**Composizione**

- Il tutto non può esistere senza le sue parti
- Se si distrugge il tutto, si distruggono anche le parti
- Ogni istanza della classe “parte” può essere componente di una sola istanza della classe “tutto”

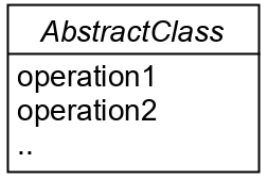



N.B. La molteplicità in corrispondenza del tutto può essere solo 0..1 o 1

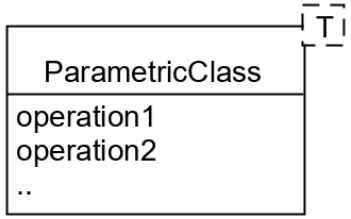
# Class Diagram: riepilogo



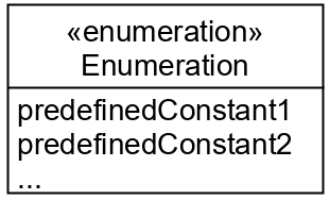

**Classe**



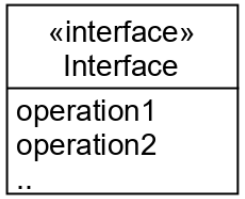

**Classe Astratta**



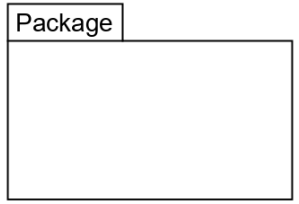

**Classe Parametrica**



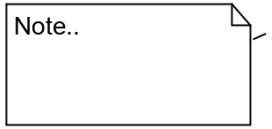

**Enumerazione**




**Interfaccia**

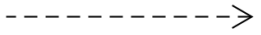



**Package**



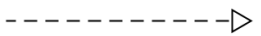

**Commento**

**Associazione**

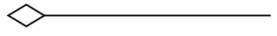


**Dipendenza**

**Generalizzazione**



**Realizzazione**



**Aggregazione**

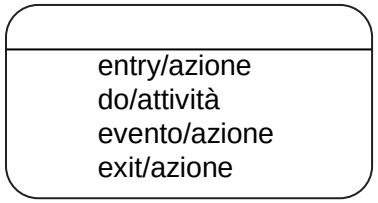
**Composizione**

# Statechart Diagram: elementi

I *diagrammi di stato (statechart diagram)* permettono di descrivere il comportamento dinamico di un oggetto o di un sistema.

Descrivono tutti gli stati raggiungibili e come cambia lo stato dell'oggetto in relazione all'accadere degli eventi (macchina a stati).

È composto da **stati** e **transizioni** fra stati.

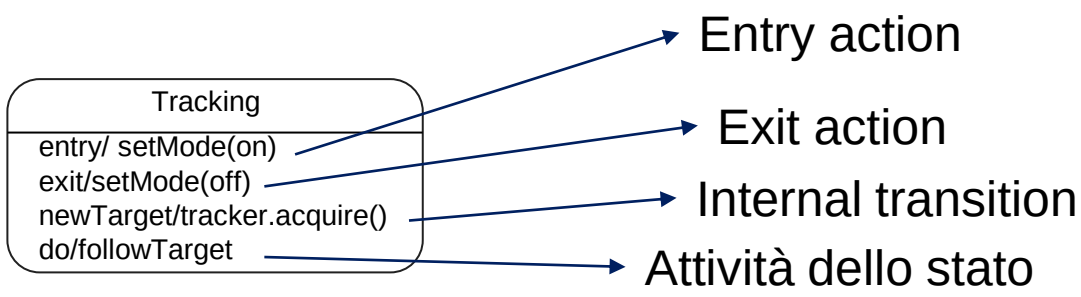



Uno **stato** è una situazione nella vita dell'oggetto in questione in cui esso soddisfa una qualche condizione, esegue una qualche attività o è in attesa di un qualche evento

- **entry action** → azione eseguita ogni volta che si entra nello stato (cioè in risposta all'evento entry)
- **exit action** → azione eseguita ogni volta che si lascia lo stato (cioè in risposta all'evento exit)
- **internal transition** → transizione che gestisce un evento senza cambiare stato (sono diverse dalle self-transition in cui si esce dallo stato rientrando nello stesso in quanto nelle self-transition si eseguono le eventuali exit e entry action)

# Statechart Diagram: elementi

Es.




---




Una transizione è una relazione che lega uno stato di partenza ed uno stato di arrivo
Ogni transizione è etichettata con tre elementi tutti opzionali:

```
Evento [Condizione] / Azione
```

- **Initial node** → Il nodo iniziale rappresenta il punto iniziale dello statechart diagram.
- **Final node** → Il nodo finale dello statechart diagram

# Biblioteca

# Caso di studio: Biblioteca

Si vuole progettare un sistema informatico per una biblioteca *non molto grande* che vuole passare da una gestione manuale a una assistita da calcolatore.

L'applicazione informatica dovrà servire sia al *personale della biblioteca* (prestiti, inventario, registrazione delle acquisizioni, etc.) sia agli *utenti* (reperimento volumi, ricerche, prestiti, etc.)

# Biblioteca: descrizione informale del sistema

La biblioteca è organizzata in settori

- storia, narrativa, saggistica, artigianato, etc.

Ogni settore contiene documenti di vario genere

- libri, riviste, materiale audio, etc.

Ogni documento risiede su uno scaffale opportunamente numerato.

È utile prevedere la presenza di uno scaffale per le “novità”.

# Biblioteca: descrizione informale del sistema

La biblioteca concede i documenti in prestito:

- ogni prestito può durare fino a 15 giorni
- è previsto un numero massimo di prestiti (attivi) per utente
- è possibile un singolo rinnovo a settimana
- è concesso il prestito ai soli utenti registrati.

La registrazione è effettuata su richiesta.

# Biblioteca: descrizione informale del sistema

Non tutti i documenti possono essere prestati (e.g. rari o preziosi o perché devono essere sempre disponibili).

In caso di mancata restituzione di un prestito:

1. l’utente viene sollecitato
2. al sollecito segue una ingiunzione con multa
3. a seguito della multa l’utente viene sospeso dal servizio

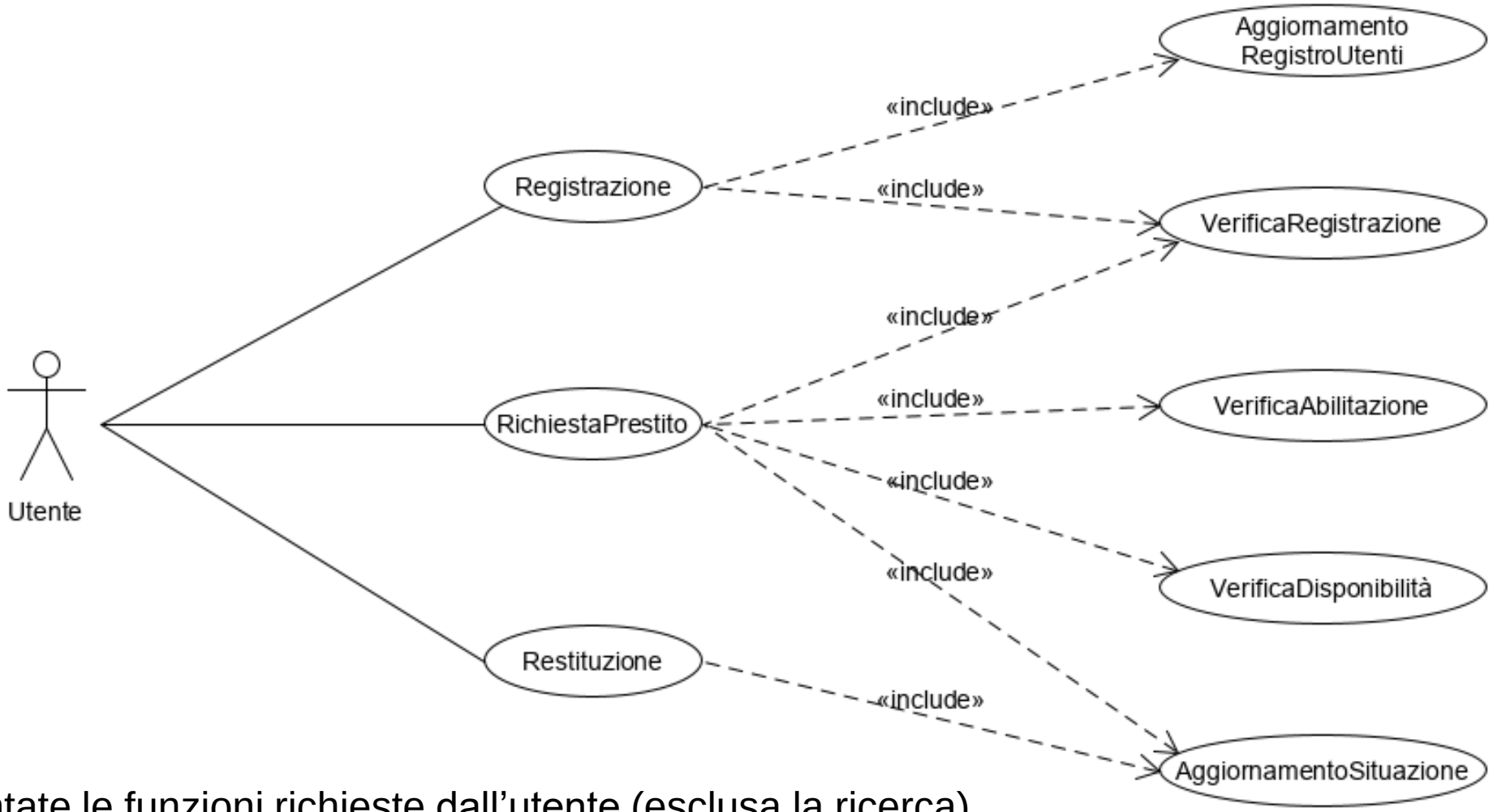
# Biblioteca: requisiti funzionali informali

Il sistema deve fornire le seguenti prestazioni:

- supporto al lavoro di amministrazione (prestiti, catalogazione documenti, operazioni di ricerca bibliografica)
- acquisto/vendita/smarrimento documenti
- gestione utenti
- gestione solleciti/multe

Relativamente alle operazioni di ricerca il sistema deve essere sufficientemente flessibile, senza essere costoso sia in termini di realizzazione che di prestazione

# Biblioteca: Use Case Diagram




Sono rappresentate le funzioni richieste dall'utente (esclusa la ricerca)

# Osservazioni sui casi d’uso

Dal diagramma non è possibile “cogliere” alcuni particolari:

- un prestito può essere concesso solo agli utenti registrati (anche se intuitivo)
- si possono/devono restituire i soli documenti presi in prestito
- non è precisato che l’aggiornamento del “registro utenti” viene eseguito solo se l’utente non è già registrato
- la procedura di registrazione non è dettagliata: è necessario essere socio/studente(?), è necessario dare una cauzione(?), etc.

Viene lasciato al lettore il compito di interpretare il significato esatto di ciascun caso d’uso.

N.B. Questo modo di procedere è pericoloso perché si presta a cattive interpretazioni dei requisiti.

# Casi d’uso in veste narrativa

È sempre opportuno corredare i diagrammi dei casi d’uso con specifiche testuali:

- Nome
- Tipo
- Precondizione
- Svolgimento normale/alternativo
- Postcondizione
- Descrizione

Non fornendo UML indicazioni specifiche a riguardo, diversi stili di descrizione possono essere adottati (svolgimenti alternativi, trigger, funzionalità correlate, etc.)

# Descrizione caso d'uso: VerificaRegistrazione

| Caso d'uso | **VerificaRegistrazione** |
|---|---|
| Tipo | Secondario |
| Precondizione | Qualche funzione sollecitata dall'utente necessita un controllo della registrazione |
| Svolgimento normale | L'utente è registrato quindi la verifica da esito positivo e fornisce l'ID utente |
| Svolgimento alternativo | L'utente non è registrato quindi la verifica ha esito negativo. Viene fornito l'esito negativo |
| Postcondizione | La registrazione è stata verificata e l'esito comunicato. Il registro utenti non ha subito variazioni |
| Descrizione | Viene eseguita quando si verificano le condizioni per effettuare un controllo della registrazione (richiesta registrazione, richiesta prestito, etc.). La verifica si basa sul contenuto del registro utenti. |

# Descrizione caso d'uso: AggiornamentoRegistroUtenti

| Caso d'uso | **AggiornamentoRegistroUtenti** |
|---|---|
| Tipo | Secondario (non è sollecitato direttamente dall'utente) |
| Precondizione | Qualche funzione sollecitata dall'utente (tipicamente la richiesta di registrazione) ha riscontrato la necessita di aggiornare il registro utenti, tipicamente aggiungendo un nuovo utente |
| Svolgimento normale | Un utente non registrato deve essere aggiunto al registro utenti. Al registro viene quindi aggiunta una registrazione concernente l'utente. All'utente è associato un identificatore numerico univoco. L'utente è registrato |
| Svolgimento alternativo | Un utente comunica delle variazioni ai propri dati (ad esempio all'indirizzo). La sua registrazione viene cambiata di conseguenza. L'ID dell'utente non cambia |
| Postcondizione | Il registro è stato modificato come richiesto |
| Descrizione | Questa operazione consiste nella modifica del registro utenti per aggiungere o aggiornare le registrazioni corrispondenti agli utenti. Per il momento non è prevista l'operazione di cancellazione utente. I dati salienti dell'utente, riportati nel registro, sono: nome, cognome, codice fiscale, indirizzo |

# Descrizione caso d'uso: Registrazione

| Caso d'uso | **Registrazione** |
|---|---|
| Tipo | Primario (sollecitata direttamente dall'utente) |
| Precondizione | L'utente richiede di essere registrato |
| Svolgimento normale | L'utente non risulta registrato, quindi viene registrato. Il sistema comunica all'operatore il numero identificatore assegnato al nuovo utente della biblioteca |
| Svolgimento alternativo | L'utente risulta essere già registrato, quindi il sistema non effettua alcuna modifica, ma si limita a comunicare all'operatore il numero identificatore |
| Postcondizione | L'utente è registrato, cioè compare nel registro utenti con un identificativo univoco |
| Descrizione | Per prima cosa si verifica se l'utente è già registrato (caso d'uso VerificaRegistrazione). Se l'esito è positivo il registro utenti non viene modificato, altrimenti viene aggiornato con l'aggiunta di una nuova registrazione corrispondente al nuovo utente (caso d'uso AggiornamentoRegistroUtenti). In ogni caso il sistema comunica all'operatore il numero ID dell'utente presso la biblioteca. |

# Descrizione caso d'uso: AggiornamentoSituazione

| Caso d'uso | **AggiornamentoSituazione** |
|---|---|
| Tipo | Secondario (non è sollecitato direttamente dall'utente) |
| Precondizione | Nell'ambito di qualche funzione sollecitata dall'utente si è riscontrata la necessità di aggiornare lo stato dei documenti e/o dei prestiti. |
| Svolgimento normale | Un insieme di documenti presenti nella biblioteca è stato concesso in prestito. Viene registrato: prestito, utente e la lista dei documenti. Viene registrato che i documenti in questione non sono presenti in biblioteca né disponibili in quanto dati in prestito. |
| Svolgimento alternativo | Un insieme di documenti è stato restituito alla biblioteca. I documenti in questione sono classificati come nuovamente disponibili al prestito. Il prestito corrispondente è registrato come concluso (cosa succede per i documenti rovinati/usurati?) |
| Svolgimento alternativo | Un insieme di nuovi documenti è stato aggiunto alla biblioteca. Viene registrato che i documenti in questione sono parte della biblioteca e che sono disponibili al prestito (sempre disponibili al prestito?) |
| Svolgimento alternativo | Un insieme di documenti è stato rimosso dalla biblioteca. Viene registrato che i documenti in questione non sono più parte del patrimonio della biblioteca (quindi vanno cancellati dal sistema informatico?) |
| Postcondizione | La descrizione della biblioteca mantenuta dal sistema informatico è aggiornata rispetto alla situazione reale. |
| Descrizione | Questo caso d'uso agisce sulla descrizione della biblioteca e in particolare sull'elenco dei documenti e sull'elenco dei prestiti, in modo che tale descrizione rappresenti fedelmente quanto succede nella realtà (prestiti, restituzioni, aggiunte e alienazioni di documenti). |

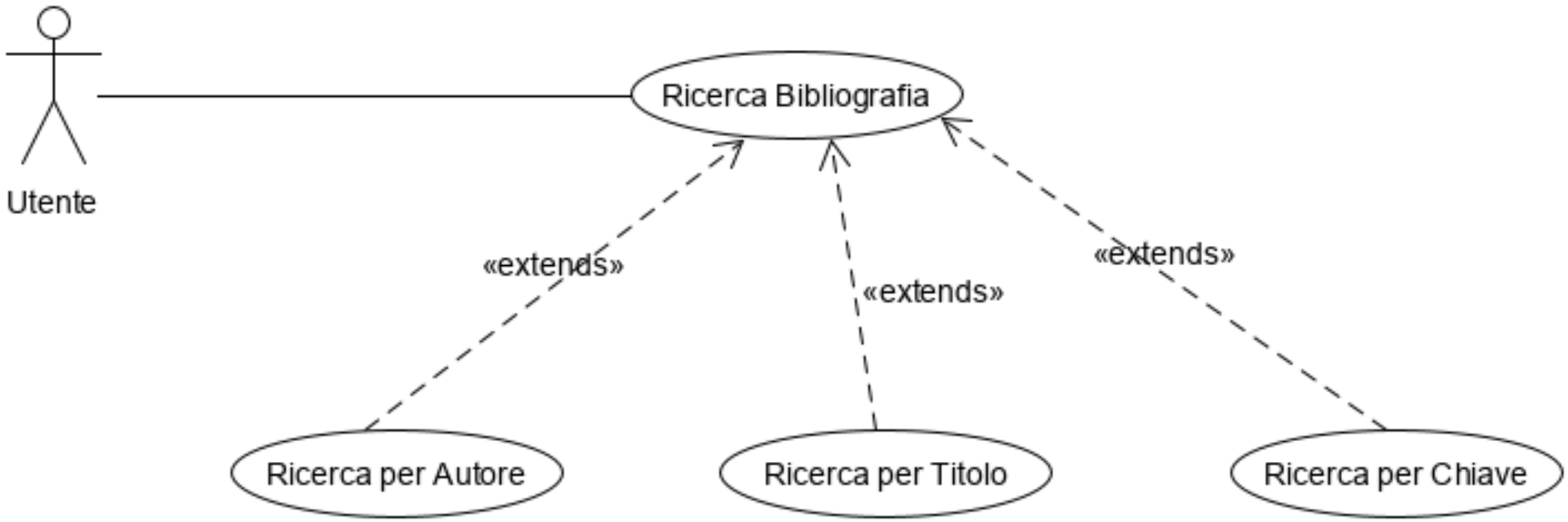
# Descrizione caso d'uso: Restituzione

| Caso d'uso | **Restituzione** |
|---|---|
| Tipo | Primario (sollecitata direttamente dall'utente) |
| Precondizione | Un insieme di documenti è stato restituito alla biblioteca |
| Svolgimento normale | Viene registrato che i documenti in questione sono nuovamente disponibili per il prestito. I prestiti corrispondenti cono classificati come conclusi. |
| Postcondizione | Lo stato del sistema è stato aggiornato in modo da rappresentare che il prestito si è concluso e il documento restituito è nuovamente presente in biblioteca e quindi disponibile |
| Descrizione | Questo caso d'uso si serve del caso d'uso secondario AggiornaSituazione per agire sulla descrizione della biblioteca e in particolare sull'elenco dei documenti e sull'elenco dei prestiti, in modo che tale descrizione rappresenti fedelmente la situazione conseguente alla restituzione. |

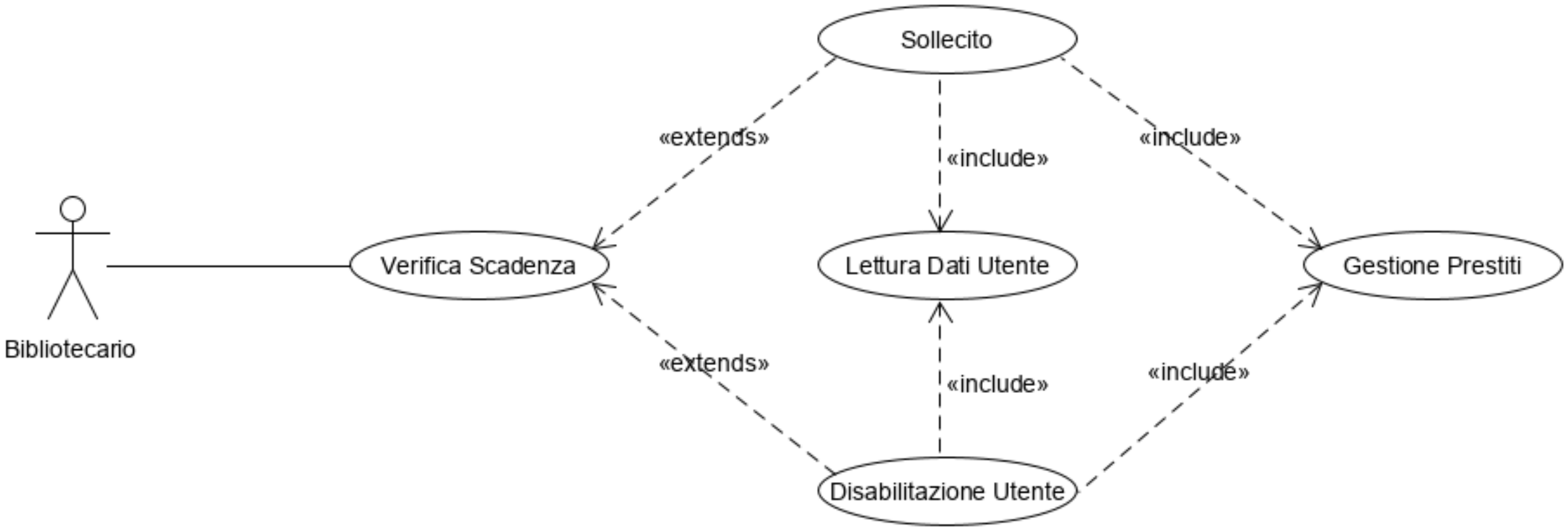
# Descrizione caso d'uso: RichiestaPrestito

| Caso d'uso | **RichiestaPrestito** |
|---|---|
| Tipo | Primario |
| Precondizione | Un utente ha richiesto un insieme di documenti |
| Svolgimento normale | L'utente registrato e abilitato ha richiesto un insieme di documenti che è possibile prendere in prestito e che sono disponibili. Il prestito regolare viene registrato tramite AggiornaSituazione |
| Svolgimento alternativo | L'utente non è registrato. Si segnala l'errore e non si modifica (lo stato) sistema |
| Svolgimento alternativo | L'utente registrato e abilitato ha richiesto documenti già in prestito o che non possono essere prestati. Si segnala l'errore e non si modifica il sistema |
| Svolgimento alternativo | L'utente registrato e abilitato ha richiesto documenti di cui alcuni non disponibili al prestito. Si segnala l'errore e si procede al prestito per quelli "disponibili" |
| Descrizione | Affinché il prestito avvenga è necessario che l'utente sia registrato e abilitato e che i documenti siano disponibili. Tali controlli sono effettuati da VerificaRegistrazione, VerificaDisponibilità, VerificaAbilitazione. Se tutte le verifiche hanno successo si procede con AggiornaSituazione. |

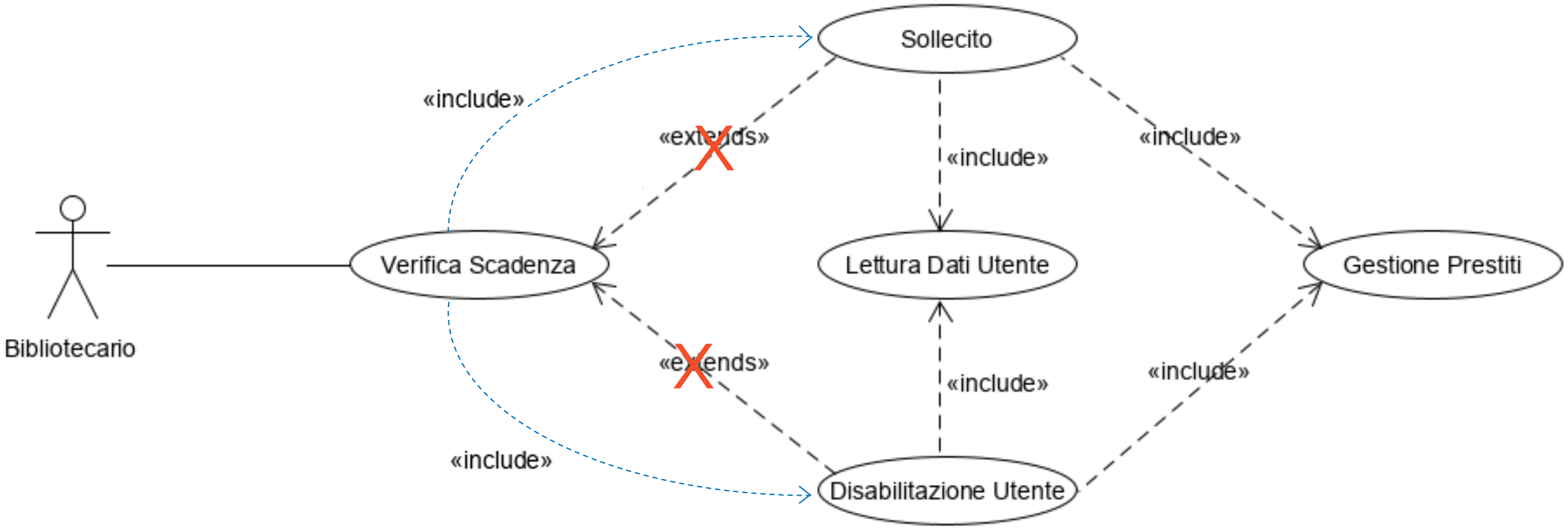
# Casi d’uso


Utente
Ricerca Bibliografia
«extends»
«extends»
«extends»
Ricerca per Autore
Ricerca per Titolo
Ricerca per Chiave

# Casi d’uso


Sollecito
«extends»
«include»
«include»
Bibliotecario
Verifica Scadenza
Lettura Dati Utente
Gestione Prestiti
«extends»
«include»
«include»
Disabilitazione Utente

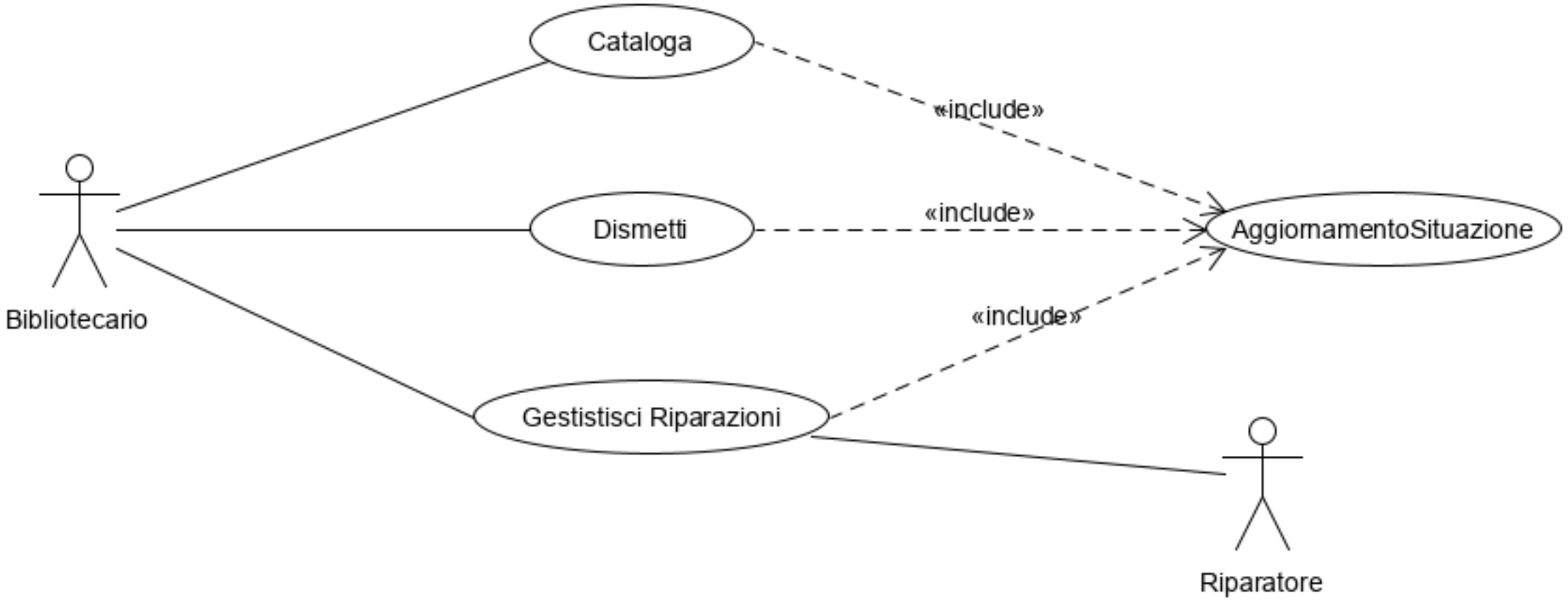
# Casi d'uso


Sollecito
«include»
«extends»
«include»
«include»
Bibliotecario
Verifica Scadenza
Lettura Dati Utente
Gestione Prestiti
«extends»
«include»
«include»
«include»
Disabilitazione Utente

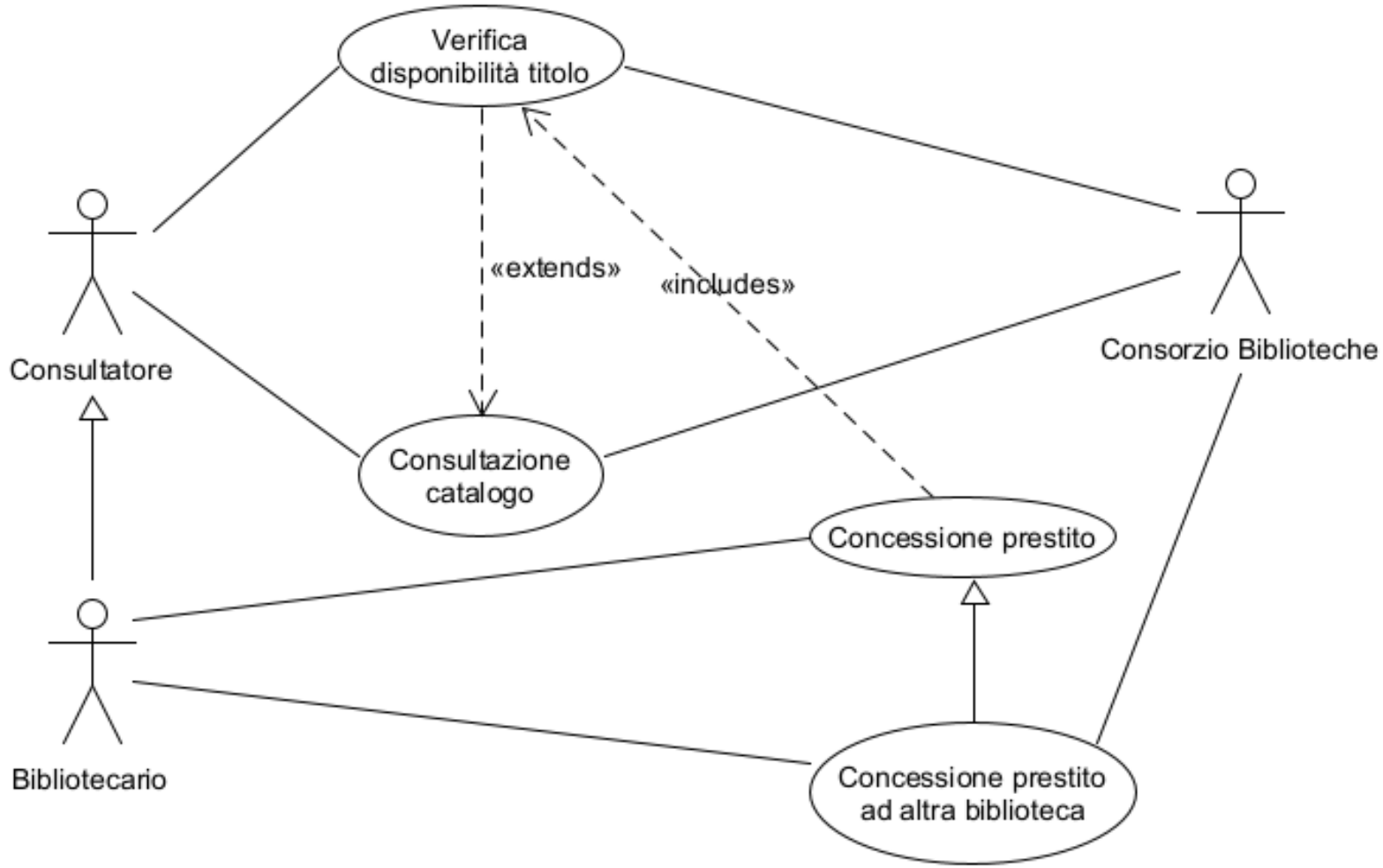
# Include vs. Extend

- Include specifica un comportamento *obbligatorio.*
- Extend specifica un comportamento *supplementare.*
- Nell'include la freccia va dal caso d'uso che include verso quello incluso.
- Nell'extend la freccia va dal caso d'uso che estende verso quello esteso.
- Sono entrambi costrutti utili, ma non bisogna abusarne altrimenti ne risente la leggibilità.

# Casi d’uso


Cataloga
«include»
Bibliotecario
Dismetti
«include»
AggiornamentoSituazione
«include»
Gestistisci Riparazioni
Riparatore

# Esempio generale utilizzo elementi e relazioni



# Descrizione dettagliata degli use-case ACTIVITY

Benché la descrizione del sistema software possa essere (in questa fase) esauriente, la descrizione del sistema nel suo complesso necessita di essere affinata.

Non si è tenuto conto di requisiti non-funzionali (e.g. prestazioni del sistema)

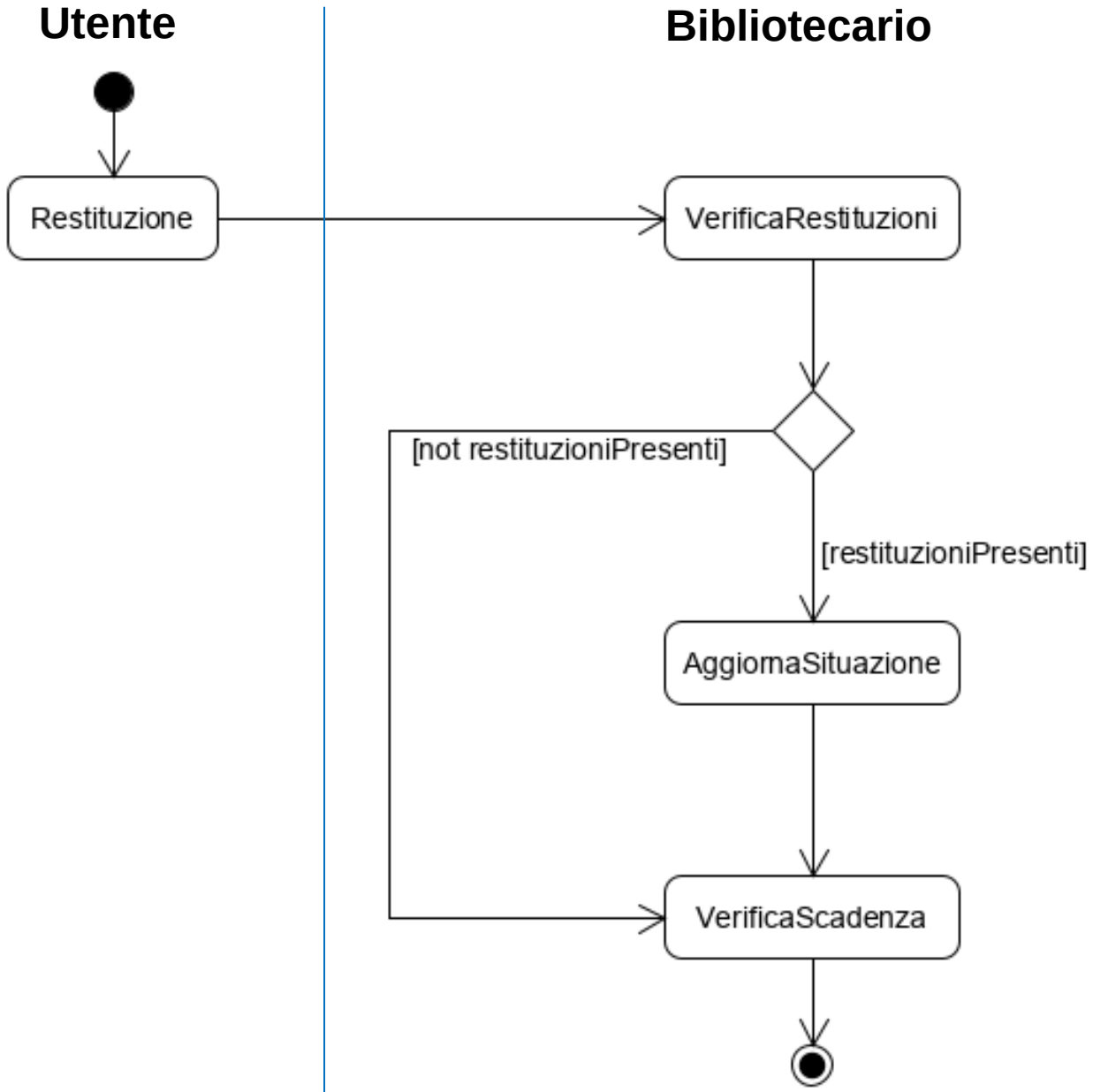
È necessario ricordare che il sistema informatico interagisce con persone (chi esegue la registrazione di un nuovo utente?)

Si consideri il processo relativo alla gestione prestiti:

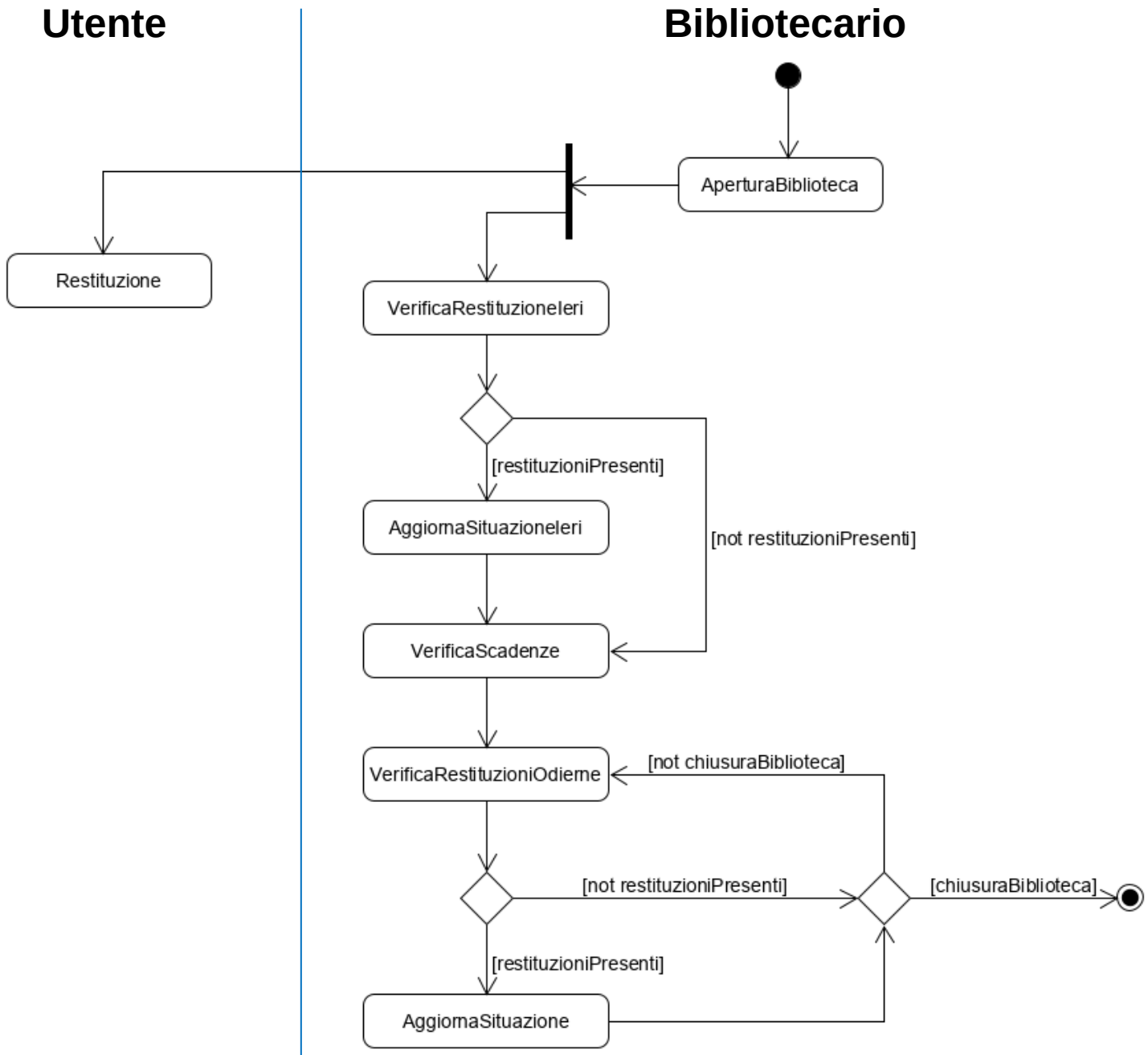
1. l'utente riporta il libro che aveva in prestito
2. il bibliotecario registra l'avvenuta restituzione
3. il bibliotecario verifica la scadenza dei prestiti

- Cosa succede se il bibliotecario non considera una restituzione?
- Cosa succede se semplicemente si invertono i punti 2 e 3?

# Descrizione activity: Restituzione/ ControlloPrestiti



# Descrizione activity: Giornata Biblioteca



# Descrizione dettagliata degli scenari SEQUENCE

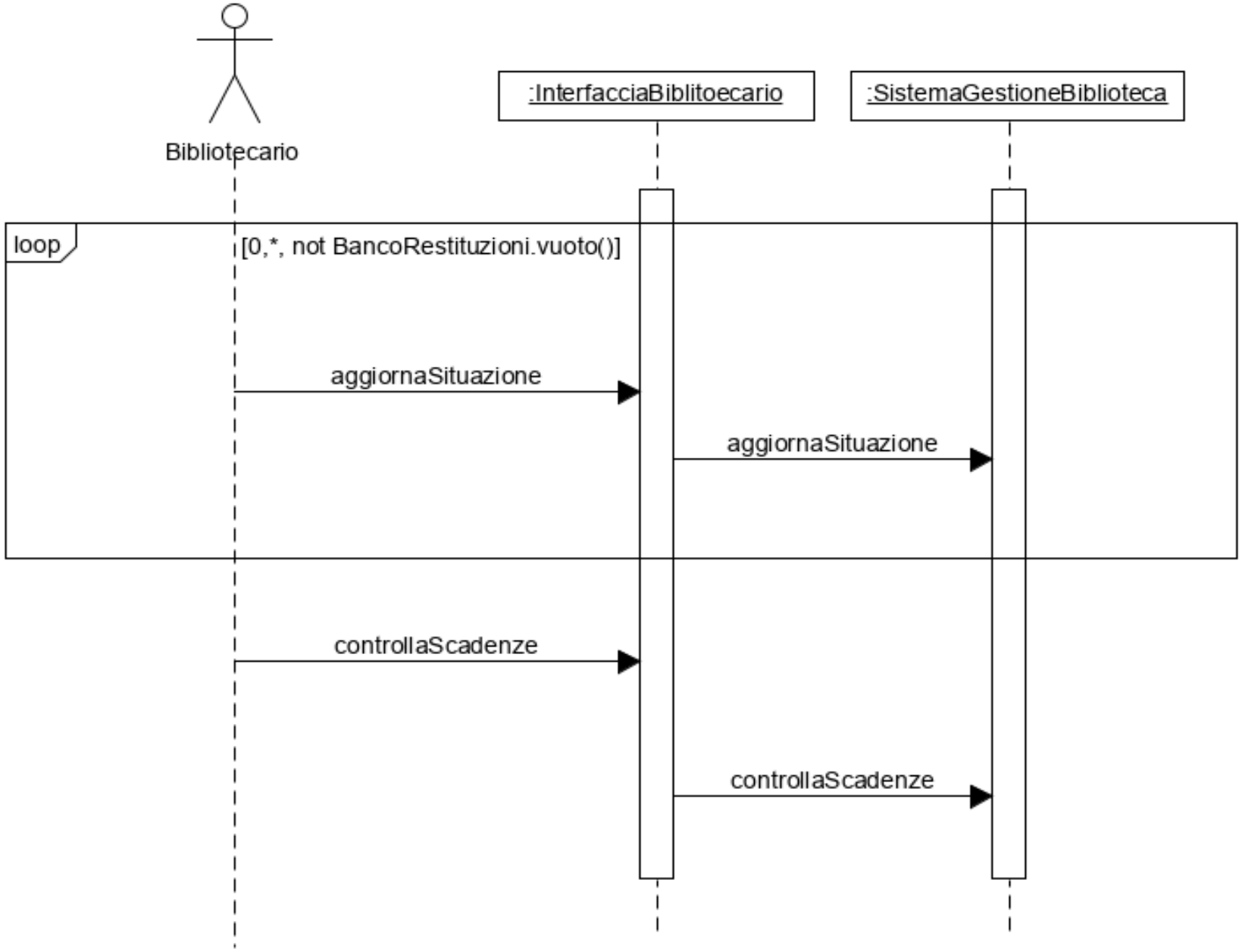
Un caso d'uso può anche essere specificato descrivendo le sue istanze o "scenari"

- uno *scenario* esemplifica il caso d'uso, descrivendolo in una sua particolare occorrenza
- a uno stesso caso d'uso possono corrispondere più scenari diversi (ad es. il sistema si comporta in modo diverso a seconda che un libro richiesto sia già in prestito oppure no)
- gli *scenari* possono essere descritti mediante *sequence diagram* (diagrammi di sequenza)
- uno stesso caso d'uso può, quindi, essere descritto da più sequence diagram

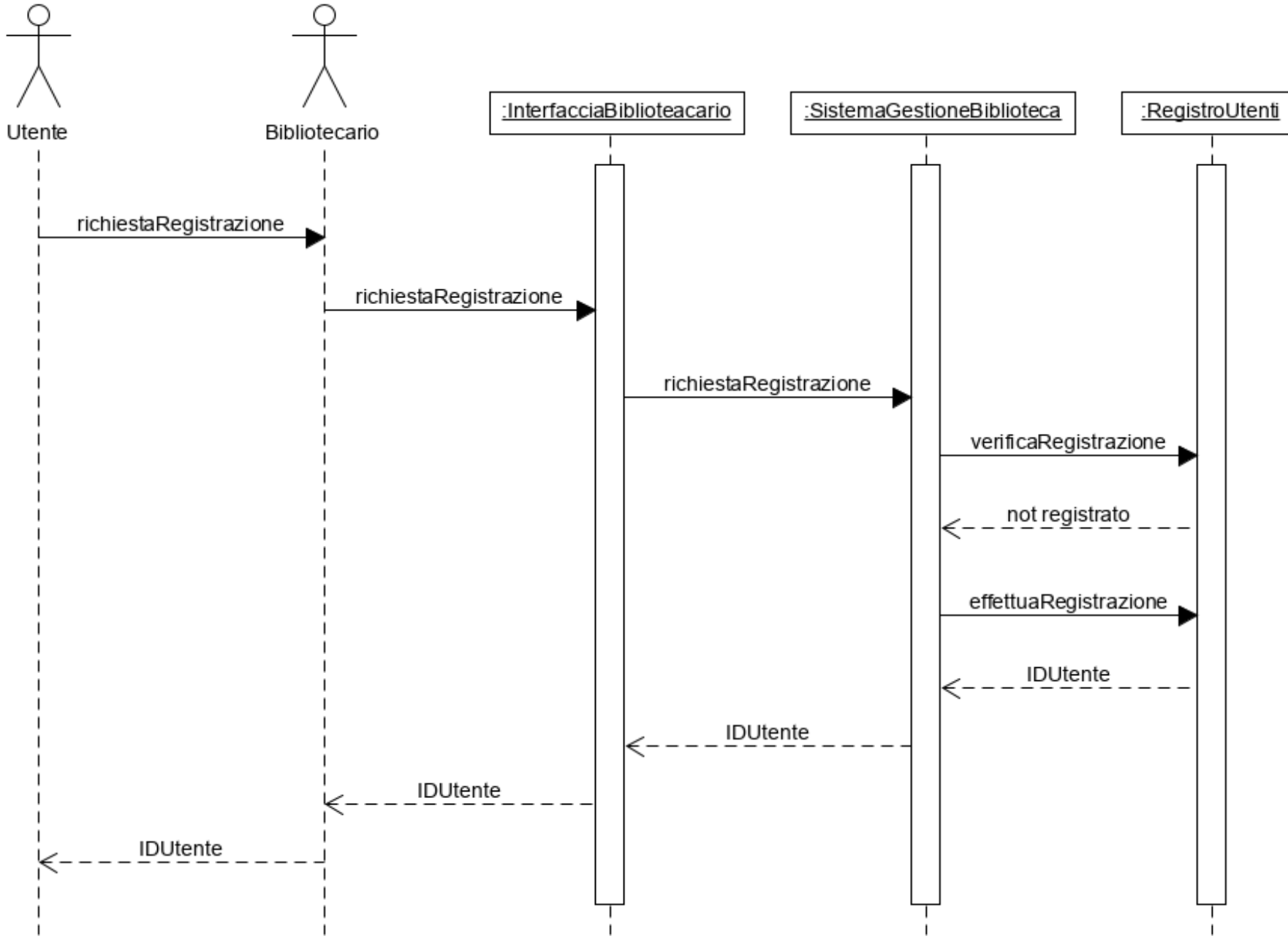
# Descrizione dettagliata degli scenari SEQUENCE

- I *sequence diagram* sono “orientati agli oggetti” (protagonisti: oggetti + messaggi)
- Si aggiungono livelli di dettaglio rispetto alla scrittura dei casi d’uso
- Gli oggetti coinvolti in uno scenario non sono in genere solo quelli del dominio informatico
- SCOPO: documentare scenari in cui l’obiettivo dell’utente non è l’uso del sistema fine a se stesso, ma il soddisfacimento di una sua esigenza, cui l’impiego dello strumento informatico è funzionale.

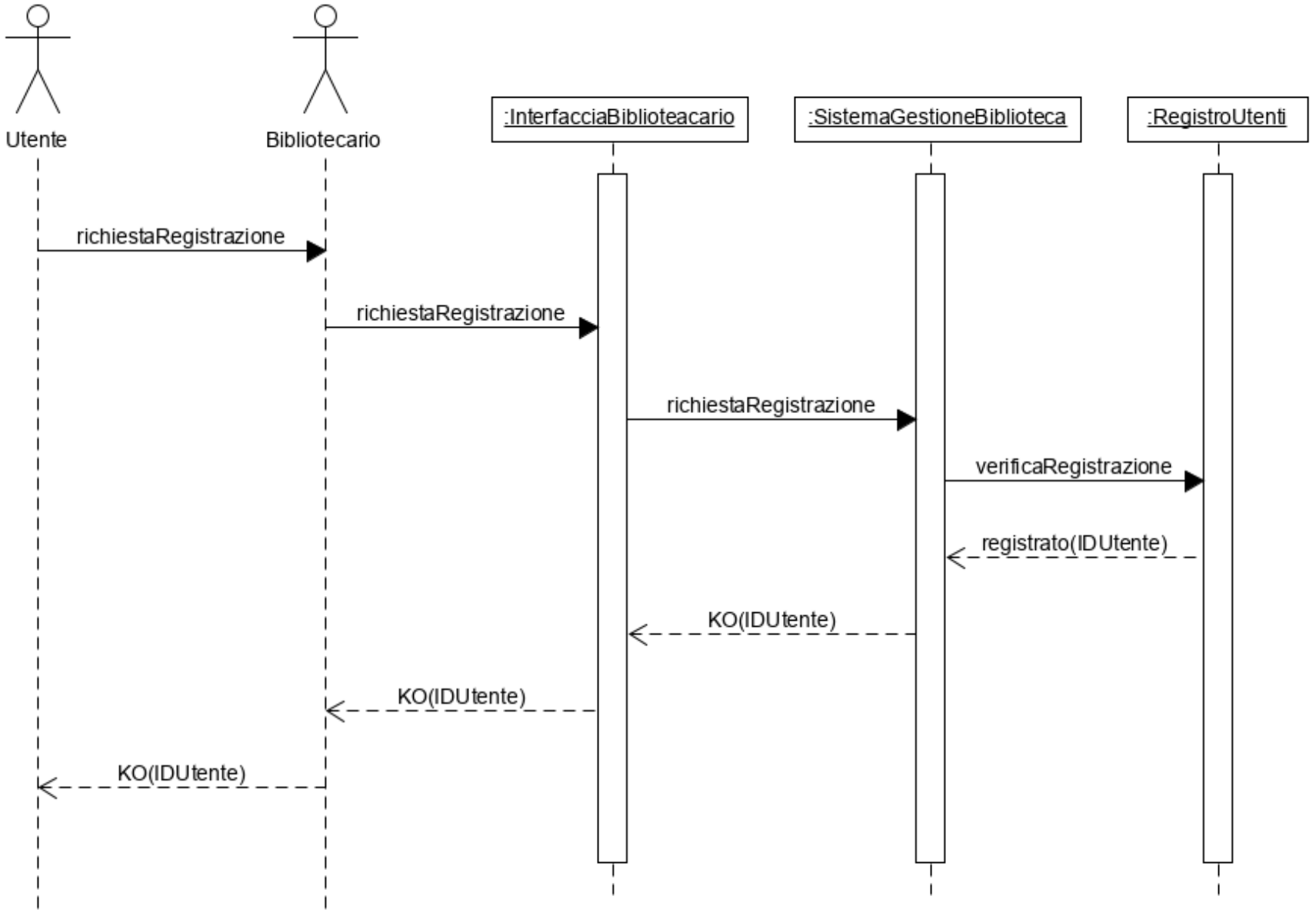
# Descrizione scenario: Restituzione/ Controllo Prestiti



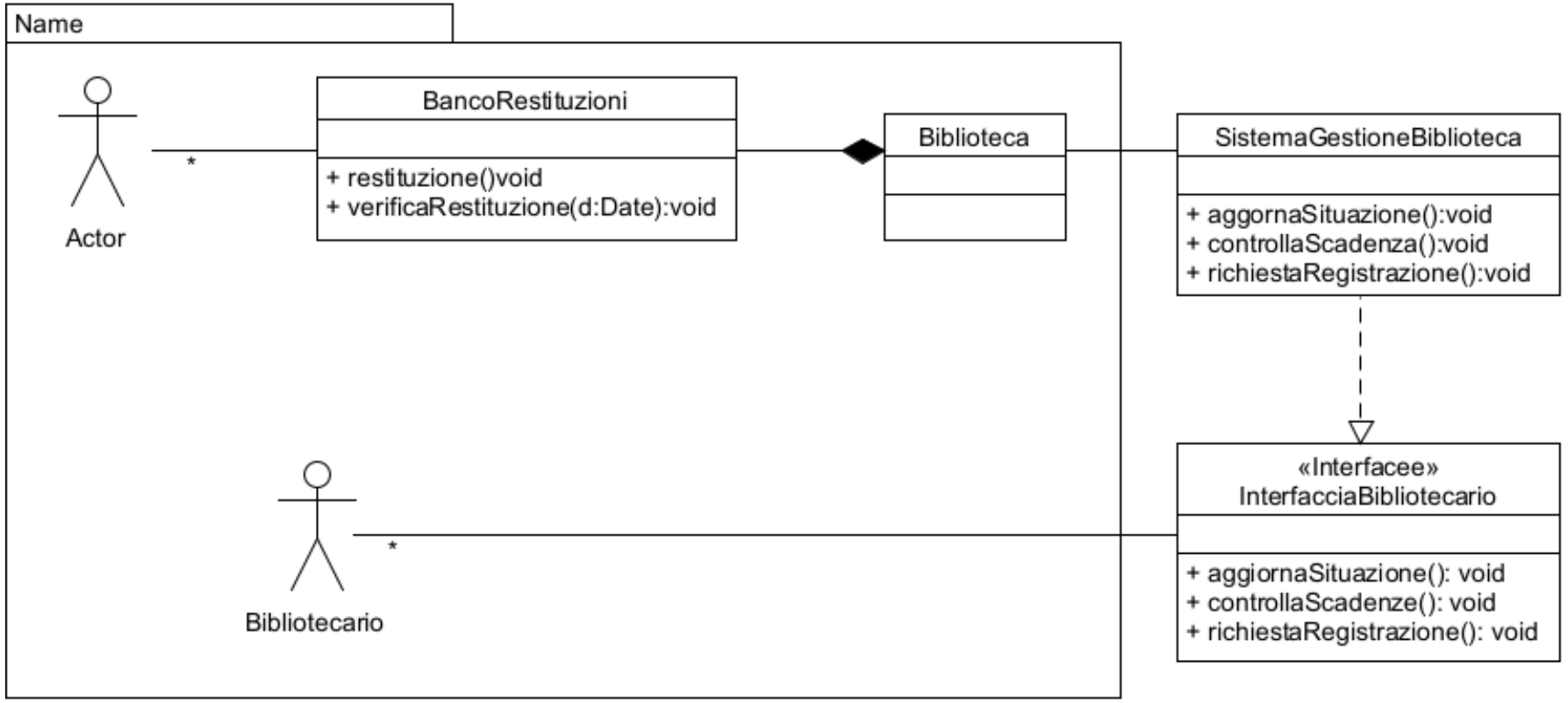
# Descrizione scenario: Registrazione Nuovo Utente



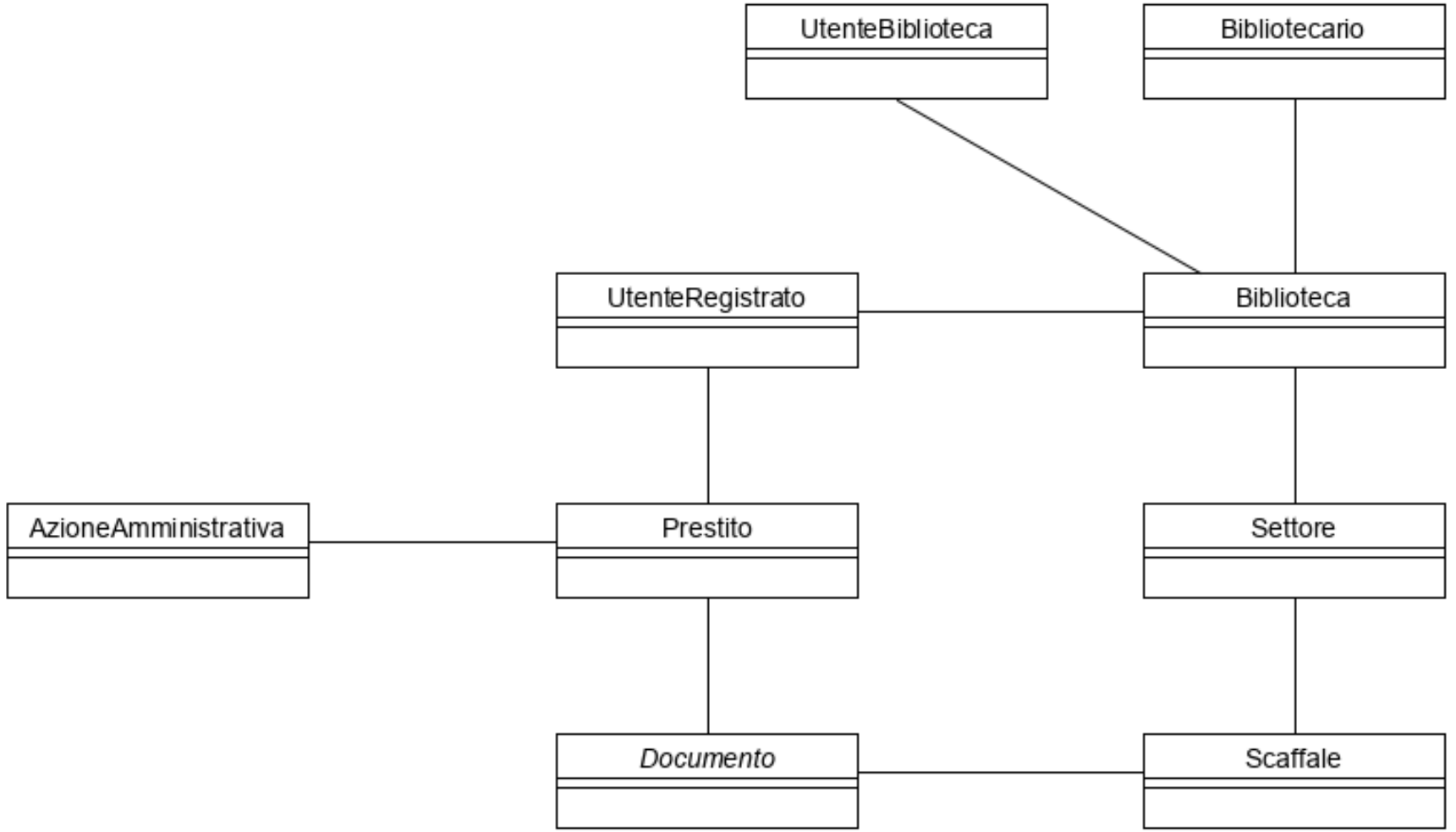
# Descrizione scenario: Registrazione Rifiutata



# Diagramma delle classi con riferimento all'ambiente esterno

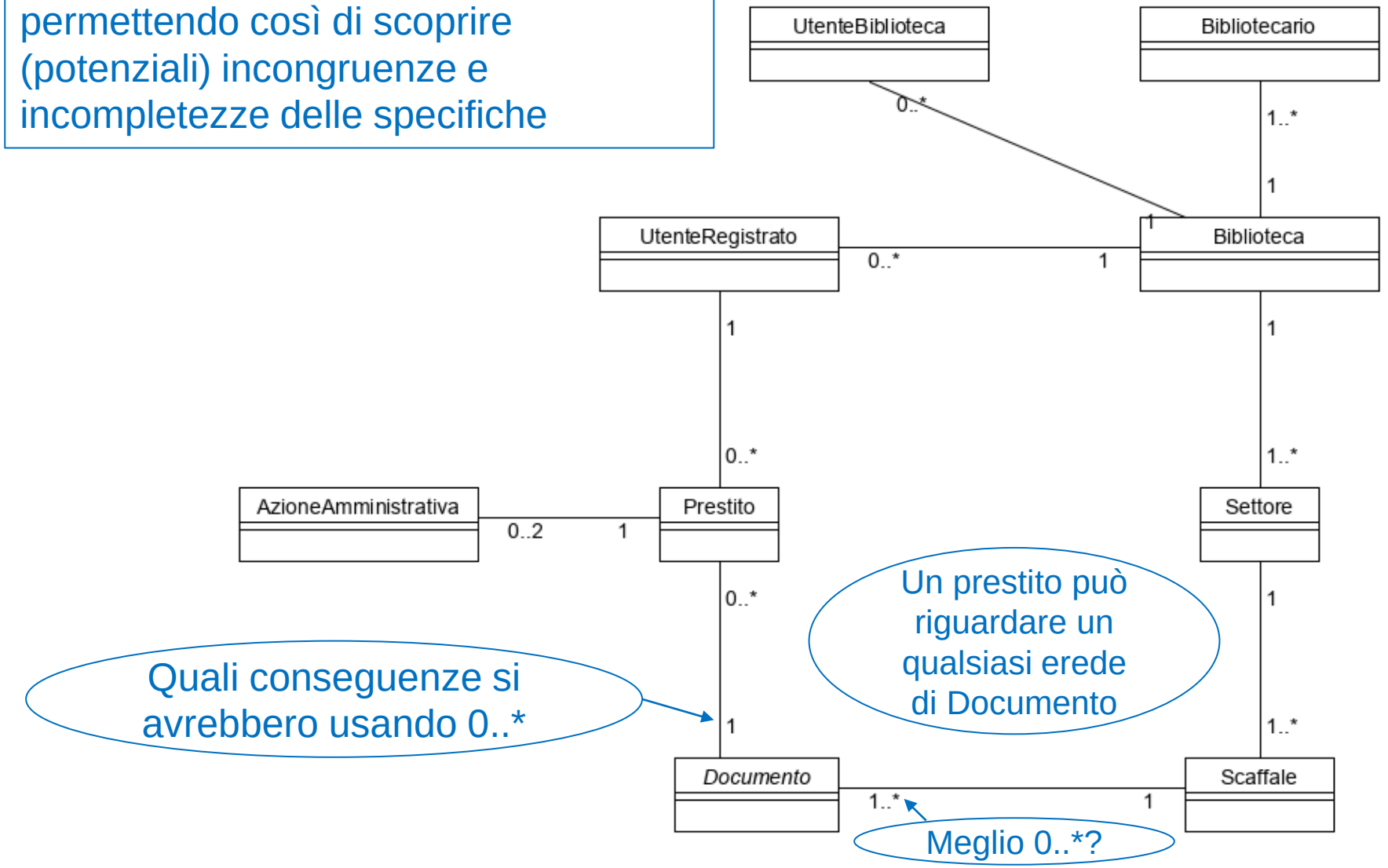


# Diagramma delle classi concettuale

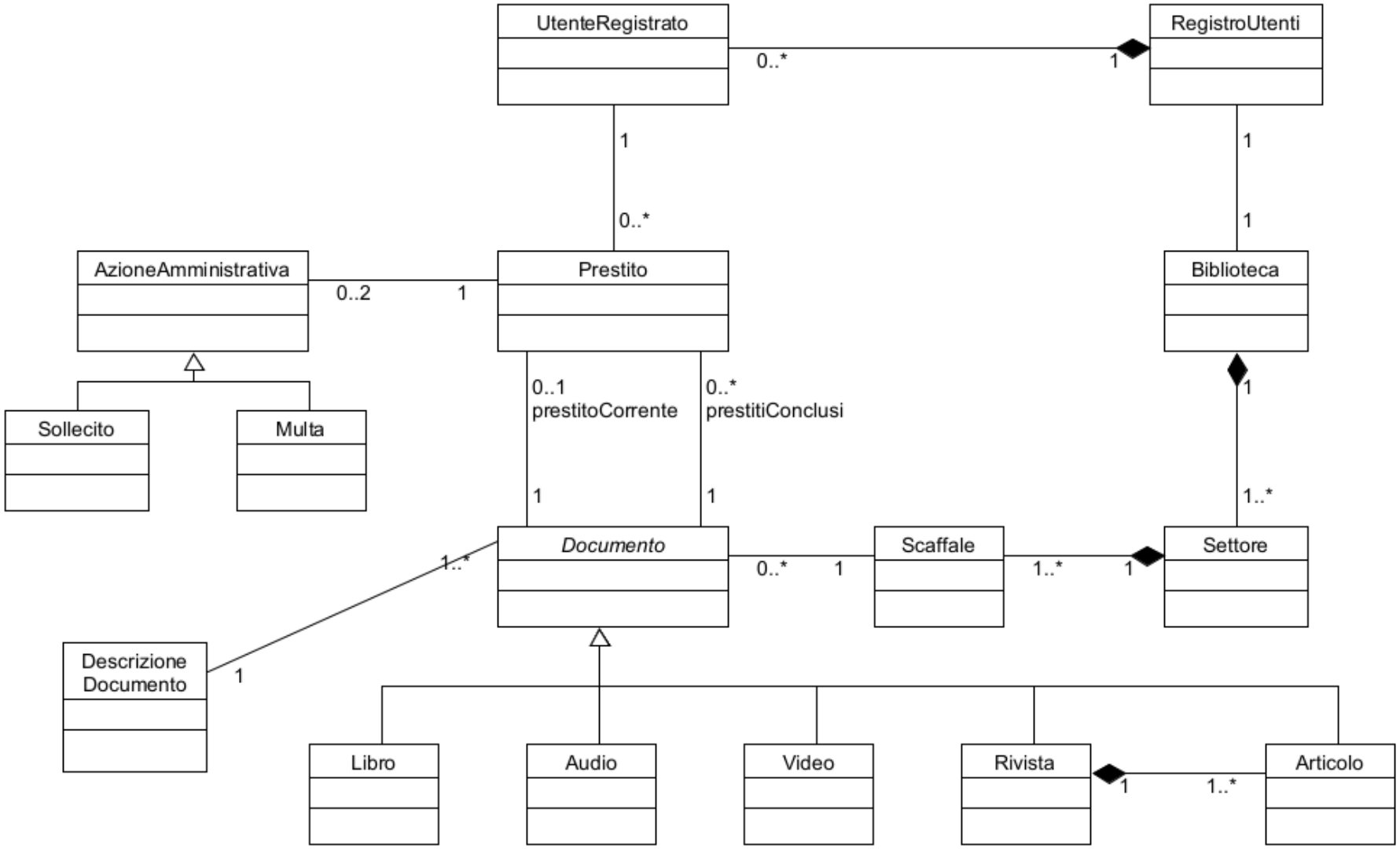


# Diagramma delle classi concettuale (cardinalità)

Specificare correttamente le relazioni e la loro cardinalità porta (solitamente) ad acquisire una conoscenza del problema più dettagliata e approfondita permettendo così di scoprire (potenziali) incongruenze e incompletezze delle specifiche



# Diagramma delle classi di progetto



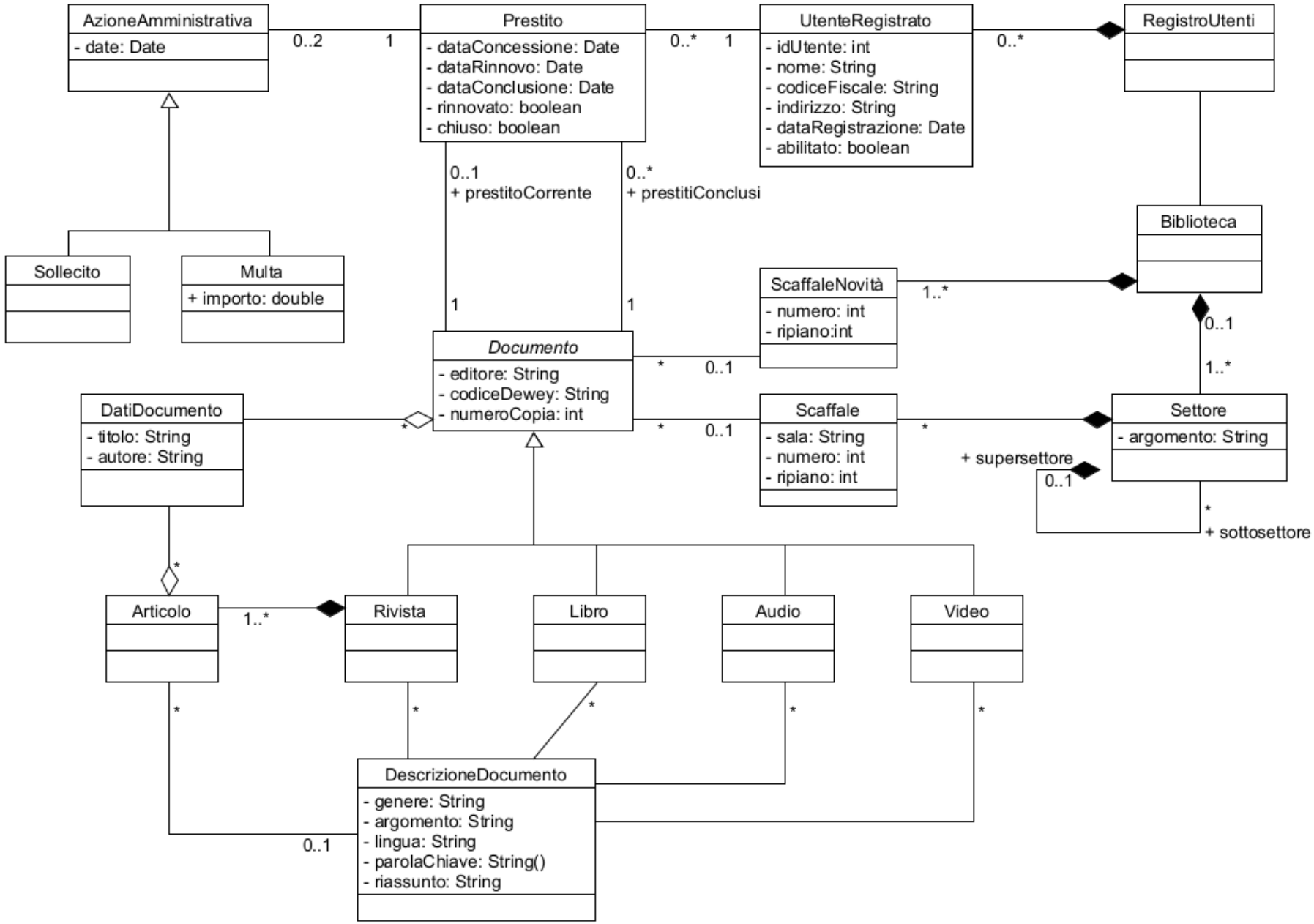
# Diagramma delle classi di progetto

Poiché il modello diventerà la base per la progettazione e successivamente per l'implementazione del sistema, è opportuno sottoporlo a periodiche valutazioni critiche.

**Osservazioni**:

- non si considera la presenza degli scaffali per le novità
- un settore può essere organizzato in sottosettori
- gli articoli ereditano la relazione con scaffale, ma questo ha senso solo con rivista
- perché non usare 0..MaxPrestiti fra UtenteRegistrato e Prestito?

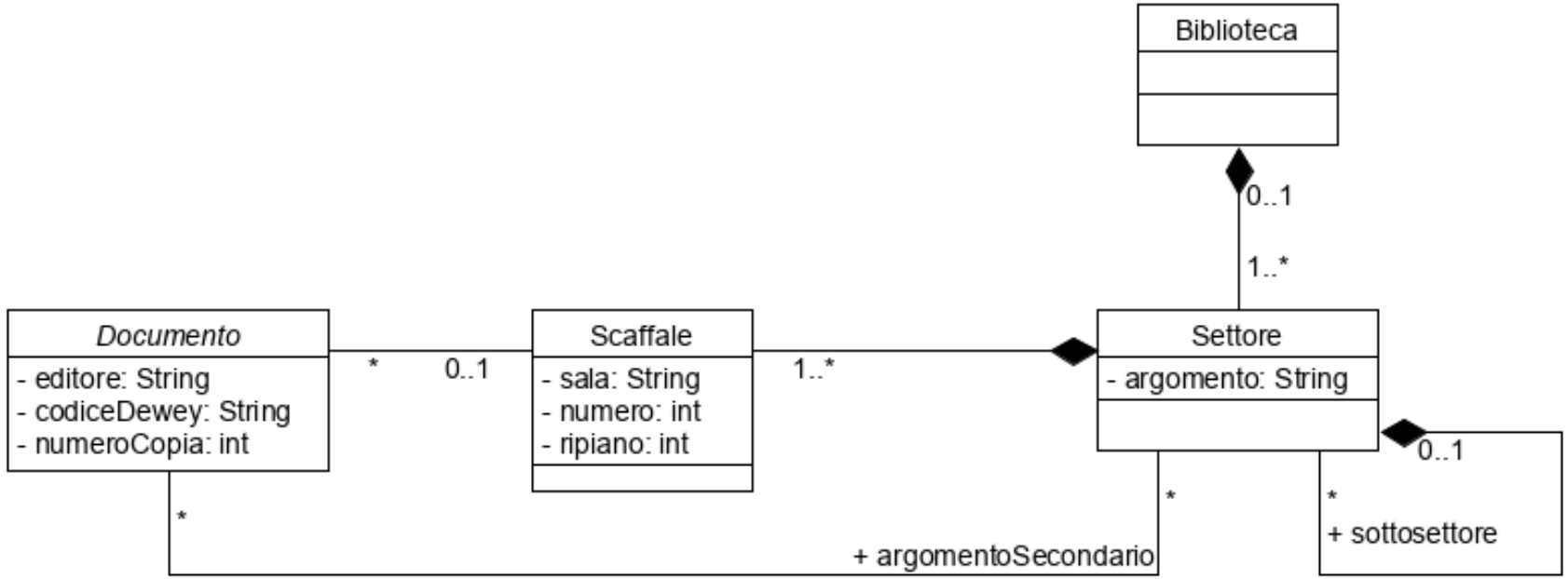
# Diagramma delle classi dettagliato



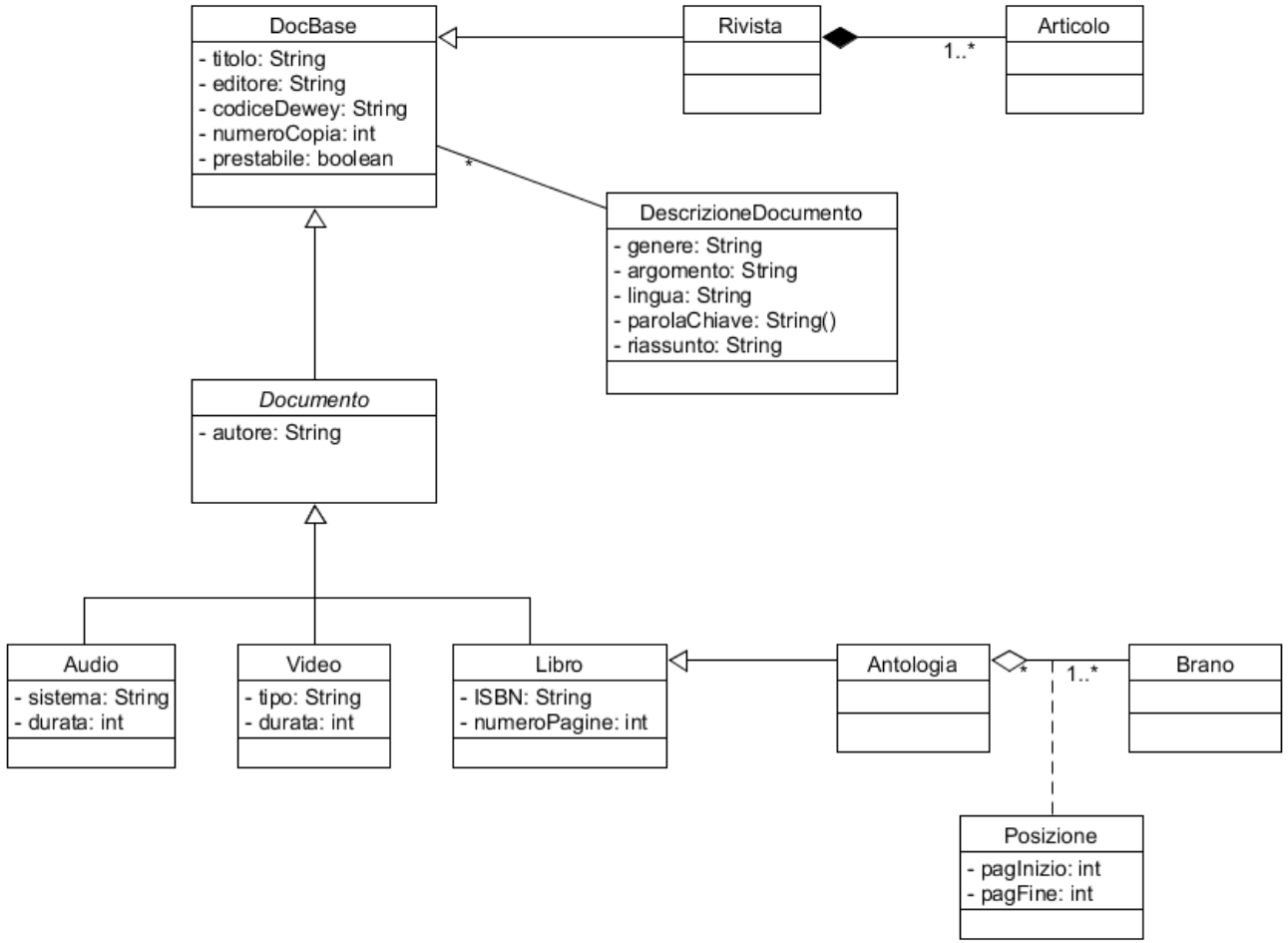
# Diagramma delle classi: affinamenti

Gestione degli argomenti secondari:

- ad esempio sicuramente esiste un libro di storia classificabile anche come di filosofia
- uno stesso documento però non può essere fisicamente collocato in due settori o/o scaffali



# Diagramma delle classi: affinamenti



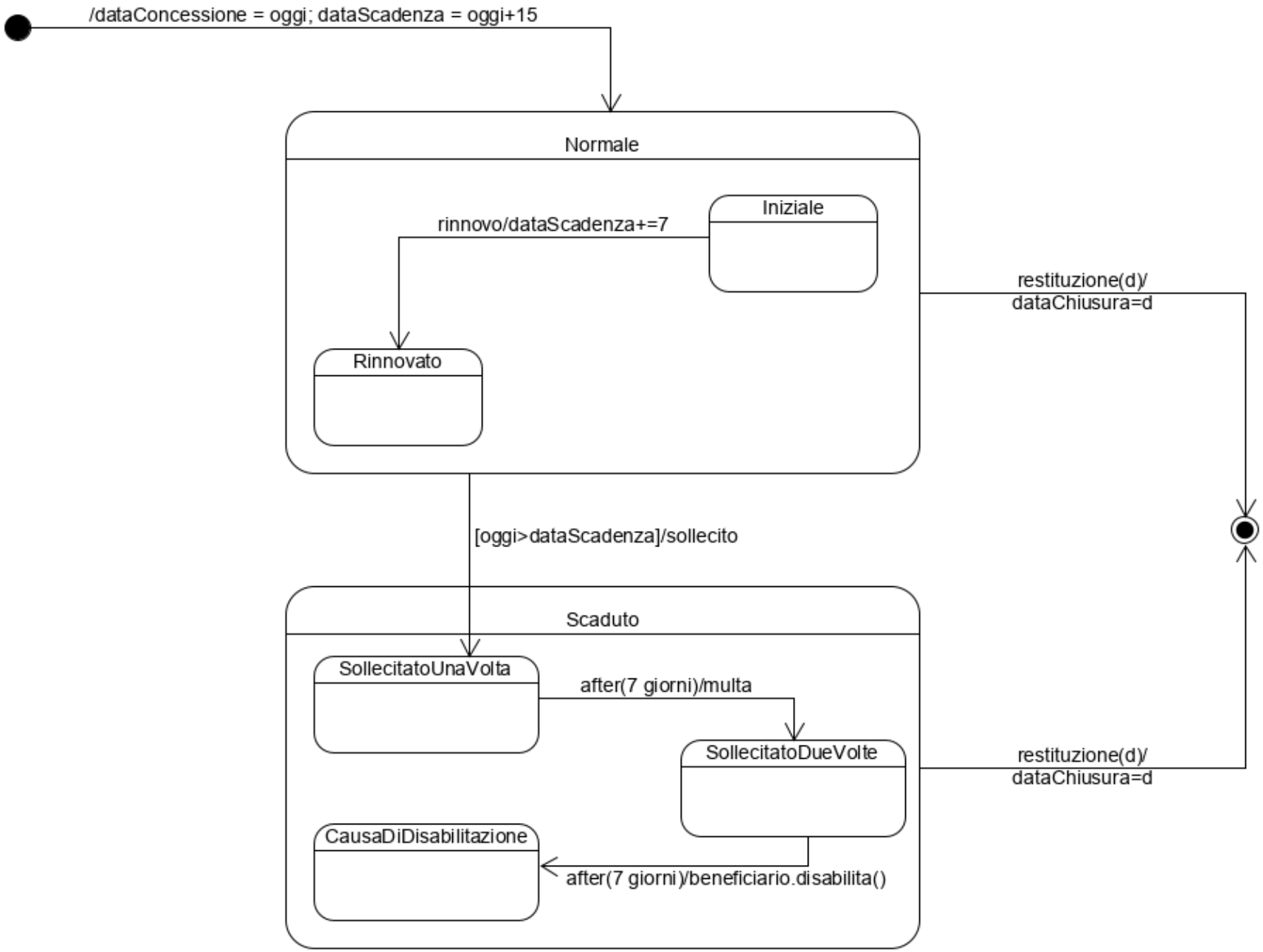

- la classe Documento è stata suddivisa in due classi (DocBase e Documento) per distinguere i documenti con autore da quelli senza
- tutti i Documenti sono associati ad una descrizione, la stessa descrizione è associata a più documenti
- la classe Rivista specializza DocBase (quindi non ha autore)
- Rivista è una composizione di Articolo (che rimane invariato)
- è introdotta Antologia
- Antologia è una specializzazione di Libro
- Antologia e Brano sono legati da una classe associativa (aggregazione)

# Analisi casi speciali

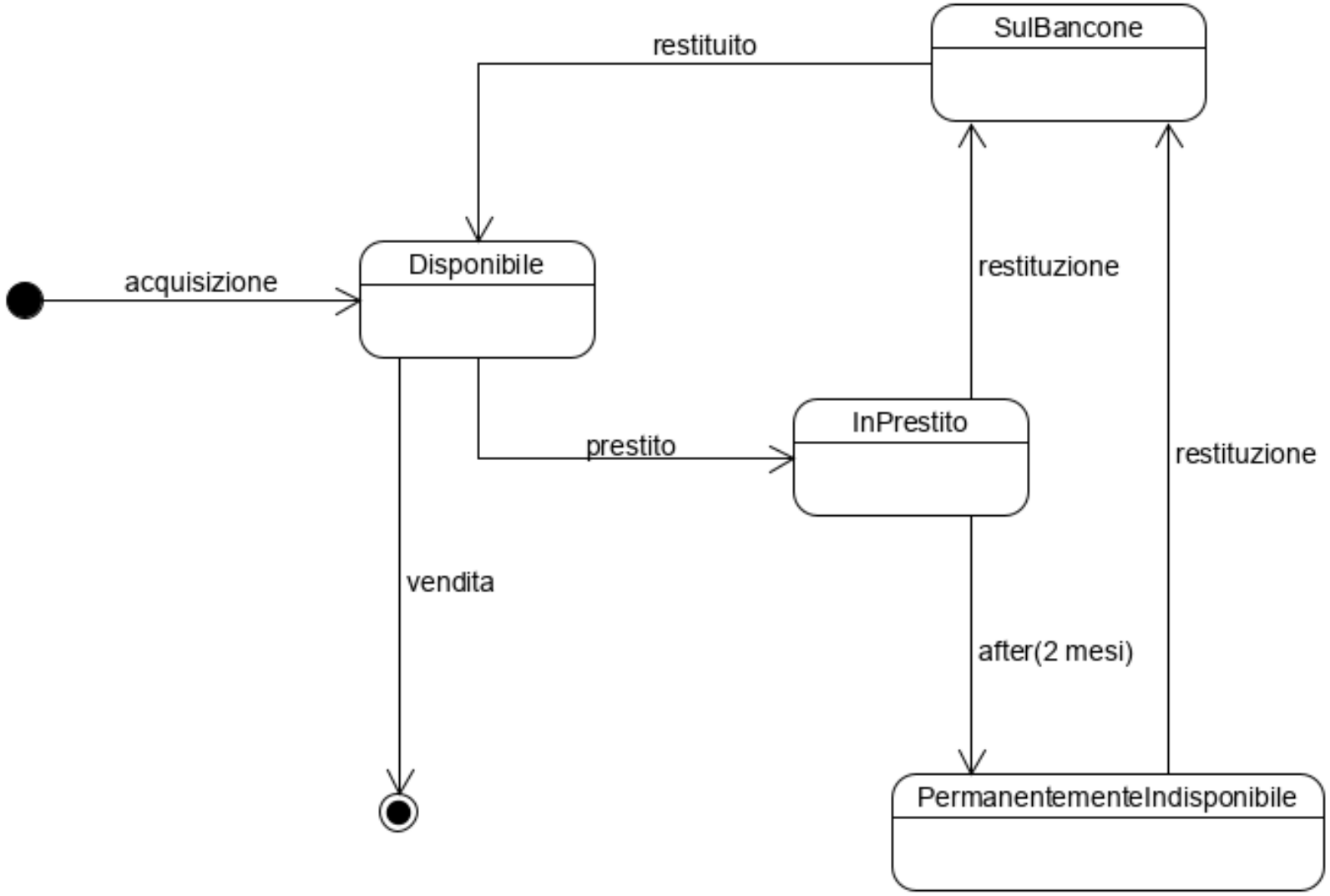
Per completare l'analisi del problema è necessario porre l'attenzione anche sui "casi speciali" o meno frequenti.

- Esistono pubblicazioni i cui autori sono istituzioni, fondazioni o altro. In questo caso è necessario aggiungere (nell'autore) informazioni quali indirizzo, telefono, etc.
- Esistono pubblicazioni con autori multipli
- Esistono documenti che hanno esistenza fisica indipendente (ad esempio libri composti da più volumi o opere registrate su più CD): come si devono modellare nel sistema(?), come gestirne il prestito(?)…
- Esistono casi (documenti) in cui l'attributo lingua potrebbe essere non rilevante oppure riduttivo (opere fruibili in più lingue).

# Diagramma degli stati: Prestito



# Diagramma degli stati: Documento



# Palestra

# Gestione Palestra

Progettare un’applicazione per la gestione di una palestra.

Il sistema deve permettere la gestione degli abbonamenti, il calcolo delle entrate annuali e mensili nonché la gestione dei clienti morosi.

# Requisiti informali

- Ad ogni **cliente** viene chiesto nome, cognome, età, indirizzo , professione e la tipologia di abbonamento e servizi richiesti
- la **palestra** offre **abbonamenti** mensili e annuali
- i clienti possono usufruire della sala pesi, della piscina, della parete da arrampicata, della sauna e del bagno turco
- sala pesi, sauna e bagno turco sono compresi nell'abbonamento base; gli altri servizi sono opzionali e sono venduti singolarmente
- a ogni cliente è associata una **scheda** per la sala pesi, che definisce gli esercizi da fare, il numero di ripetizioni e la frequenza.

# Requisiti informali

- La scheda può essere organizzata su una visita alla palestra a settimana oppure su due
- nel caso i clienti siano abbonati a servizi aggiuntivi (ad esempio la piscina), la scheda può considerare anche questo tipo di servizi e di conseguenza il numero di visite settimanali aumenta
- la palestra offre, a chi fosse interessato, uno staff medico per assistere i clienti, stabilirne il grado di forma fisica e definire diete opportune
- chi richiede questo servizio deve sottoporsi a controlli mensili (peso, massa grassa, frequenza cardiaca sotto sforzo e a riposo) e i risultati vengono opportunamente archiviati.

# Requisiti informali

Il sistema deve offrire le seguenti funzionalità:

1. elaborazione del bilancio complessivo (entrate e abbonamenti) della palestra sia mese per mese sia alla fine di un anno

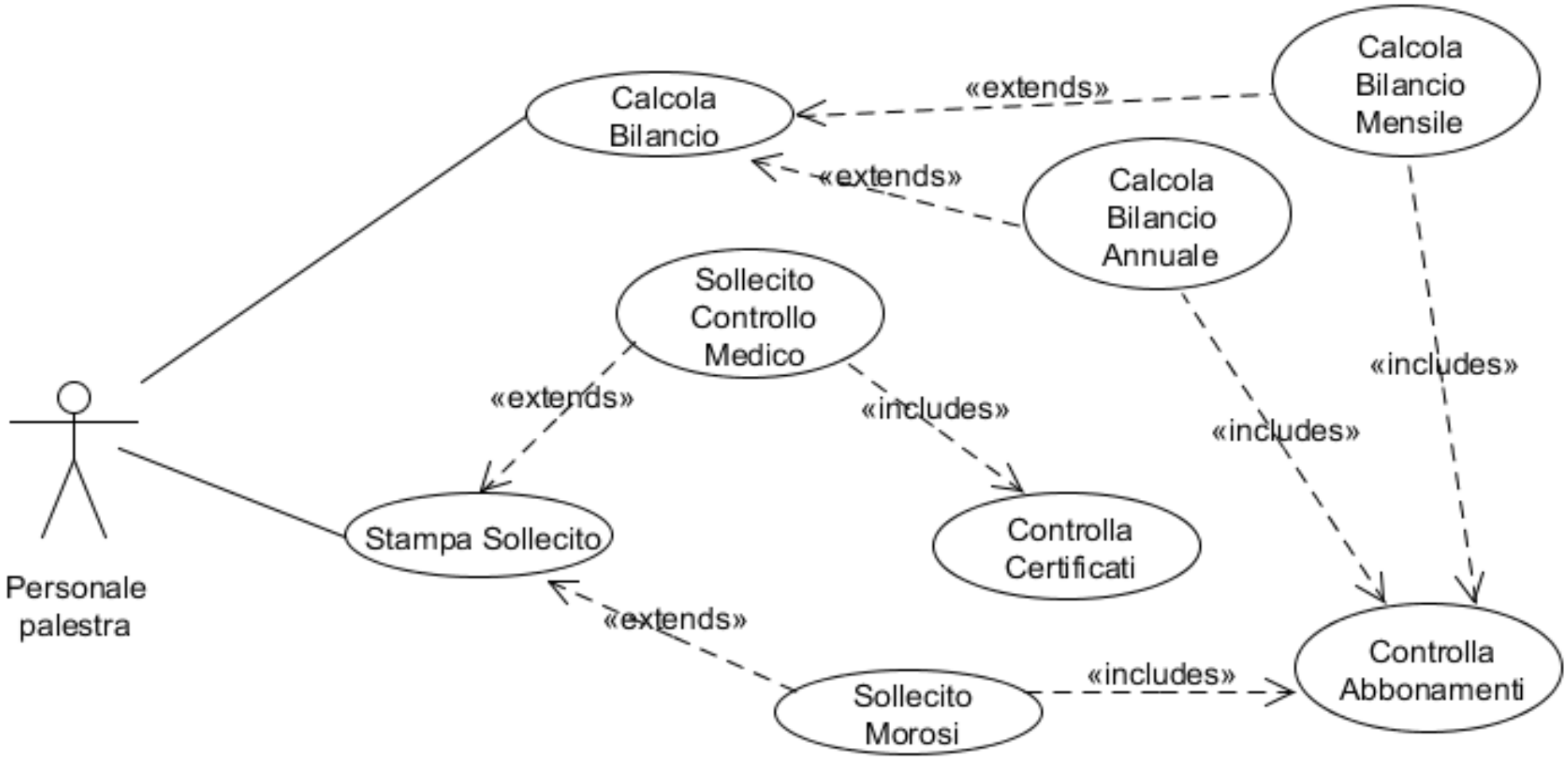
2. stampa sollecito dei clienti morosi o che devono sottoporsi al controllo mensile

3. gestione della tipologia dei servizi offerti (aggiunta, rimozione, modifica)

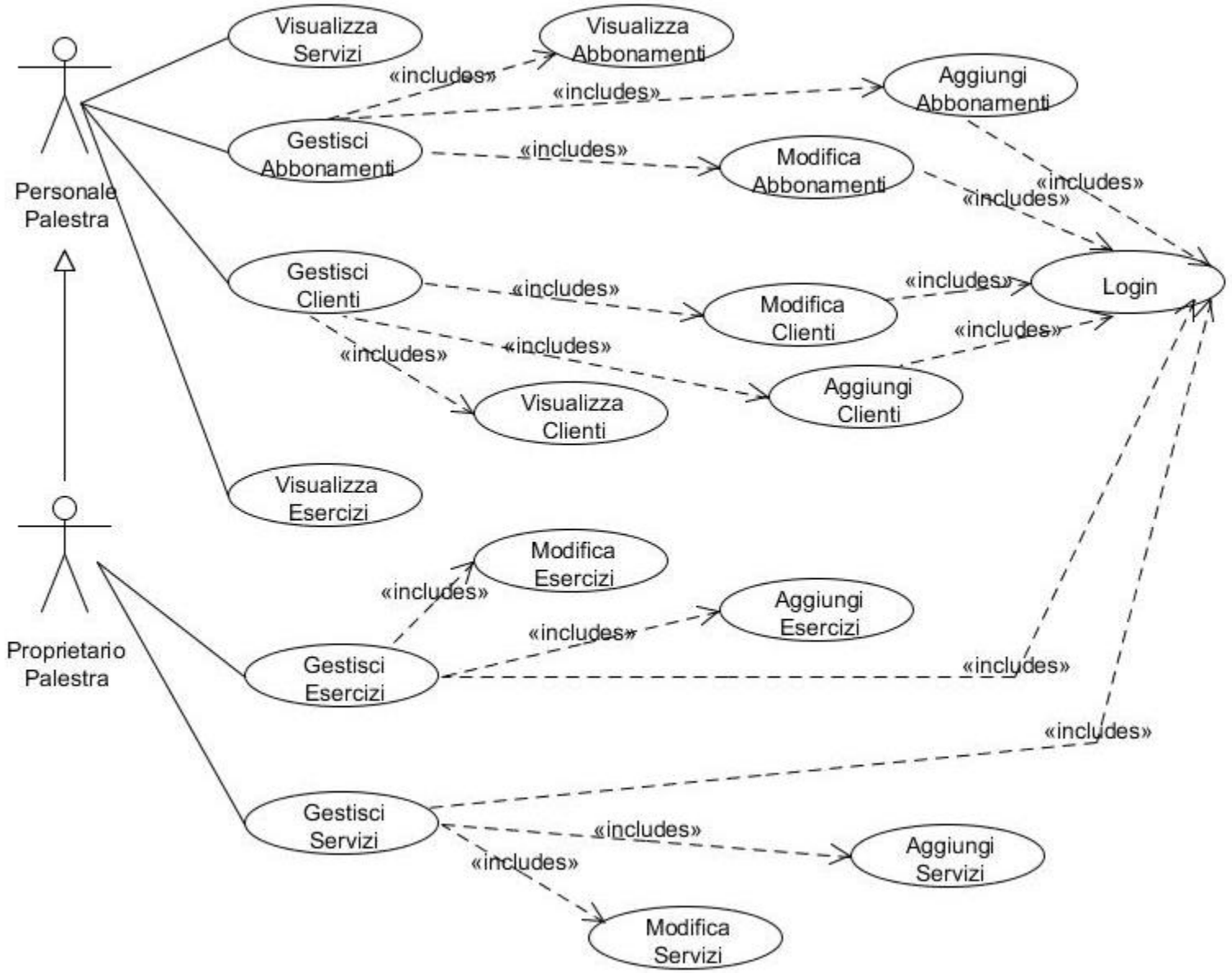
4. gestione degli abbonamenti e dell'anagrafica dei clienti (aggiunta, rimozione e verifica)

NOTA: I punti 3 e 4 sono lasciati come esercizio. Prevedere a riguardo la realizzazione di USE-CASE, SEQUENCE e valutare l'impatto delle nuove funzionalità sul diagramma delle classi riportato di seguito.

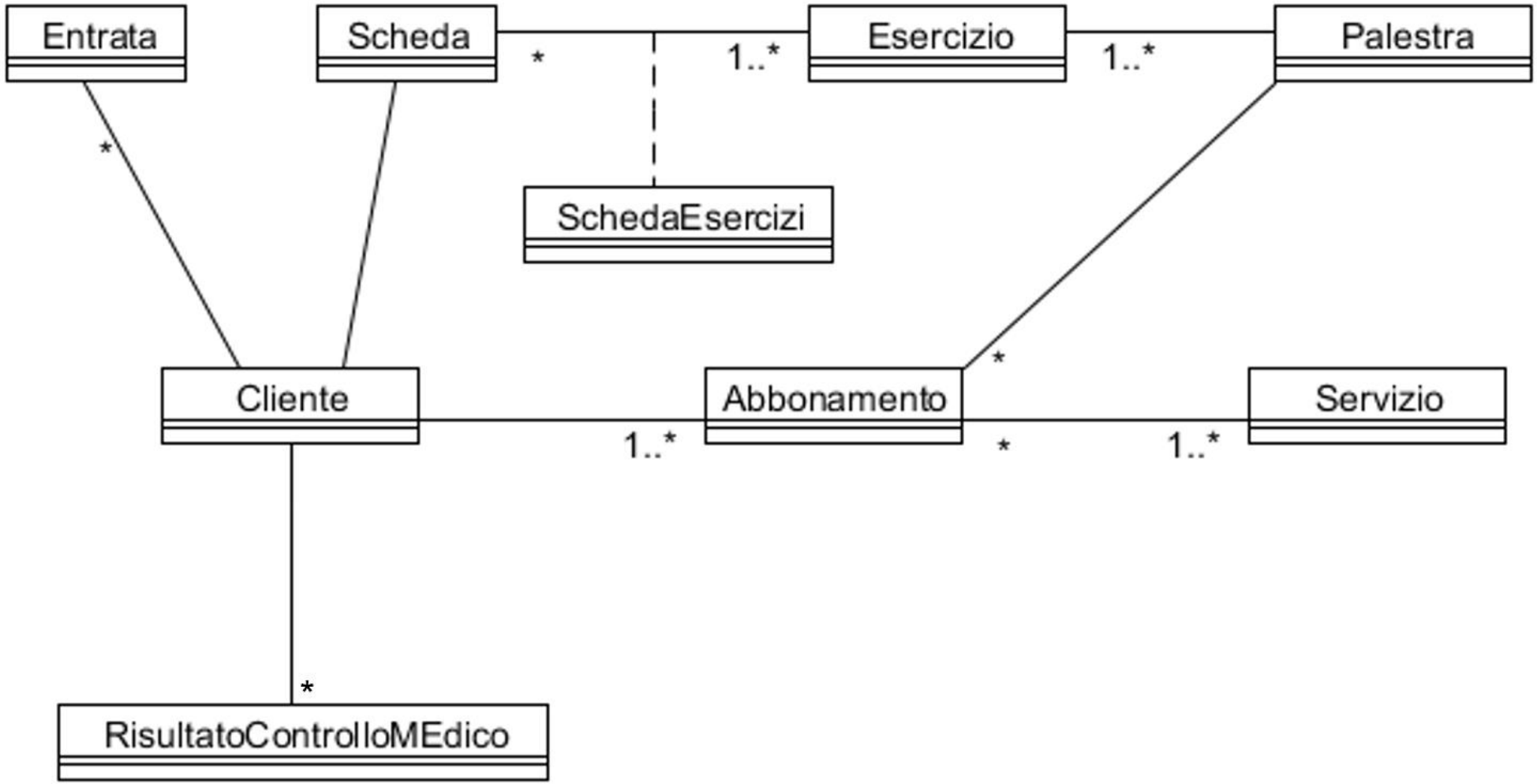
# Casi d’uso


Calcola Bilancio
«extends»
Calcola Bilancio Mensile
«extends»
Calcola Bilancio Annuale
Sollecito Controllo Medico
«extends»
«includes»
«includes»
«includes»
Stampa Sollecito
Controlla Certificati
Personale palestra
«extends»
Sollecito Morosi
«includes»
Controlla Abbonamenti

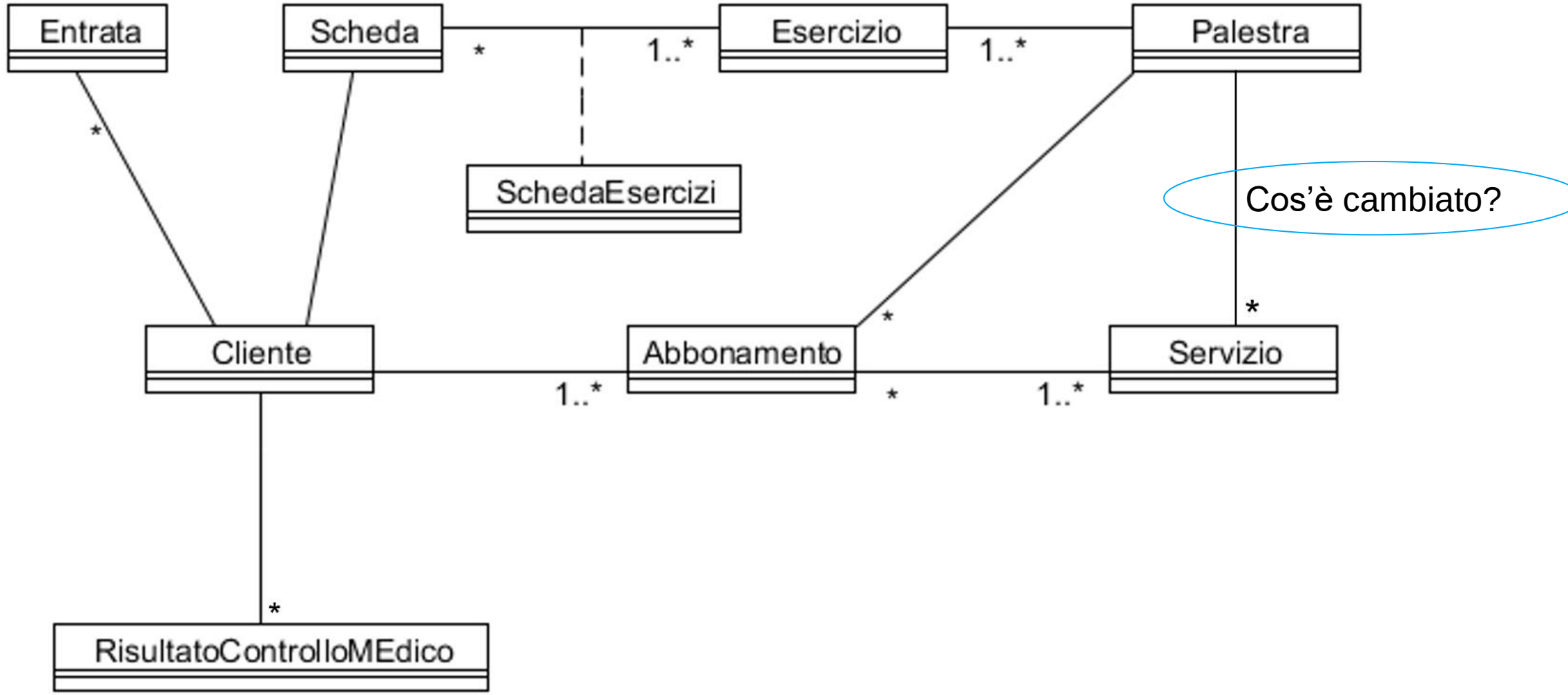
# Casi d'uso


Personale Palestra
Proprietario Palestra
Visualizza Servizi
Visualizza Abbonamenti
Aggiungi Abbonamenti
Gestisci Abbonamenti
Modifica Abbonamenti
Gestisci Clienti
Modifica Clienti
Login
Visualizza Clienti
Aggiungi Clienti
Visualizza Esercizi
Modifica Esercizi
Aggiungi Esercizi
Gestisci Esercizi
Gestisci Servizi
Aggiungi Servizi
Modifica Servizi
«includes»

# Diagramma concettuale delle classi



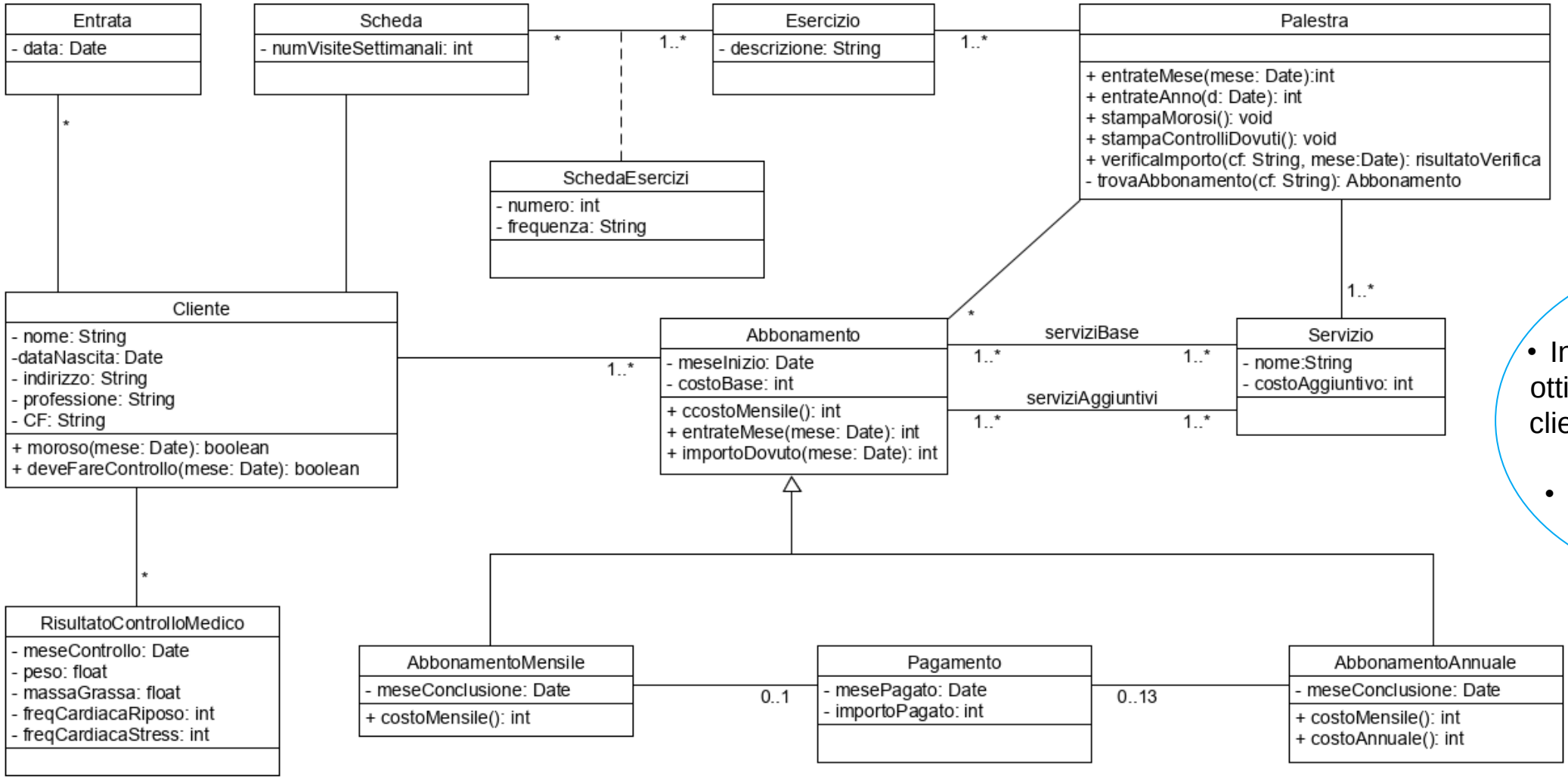
# Diagramma concettuale delle classi



# Diagramma concettuale delle classi

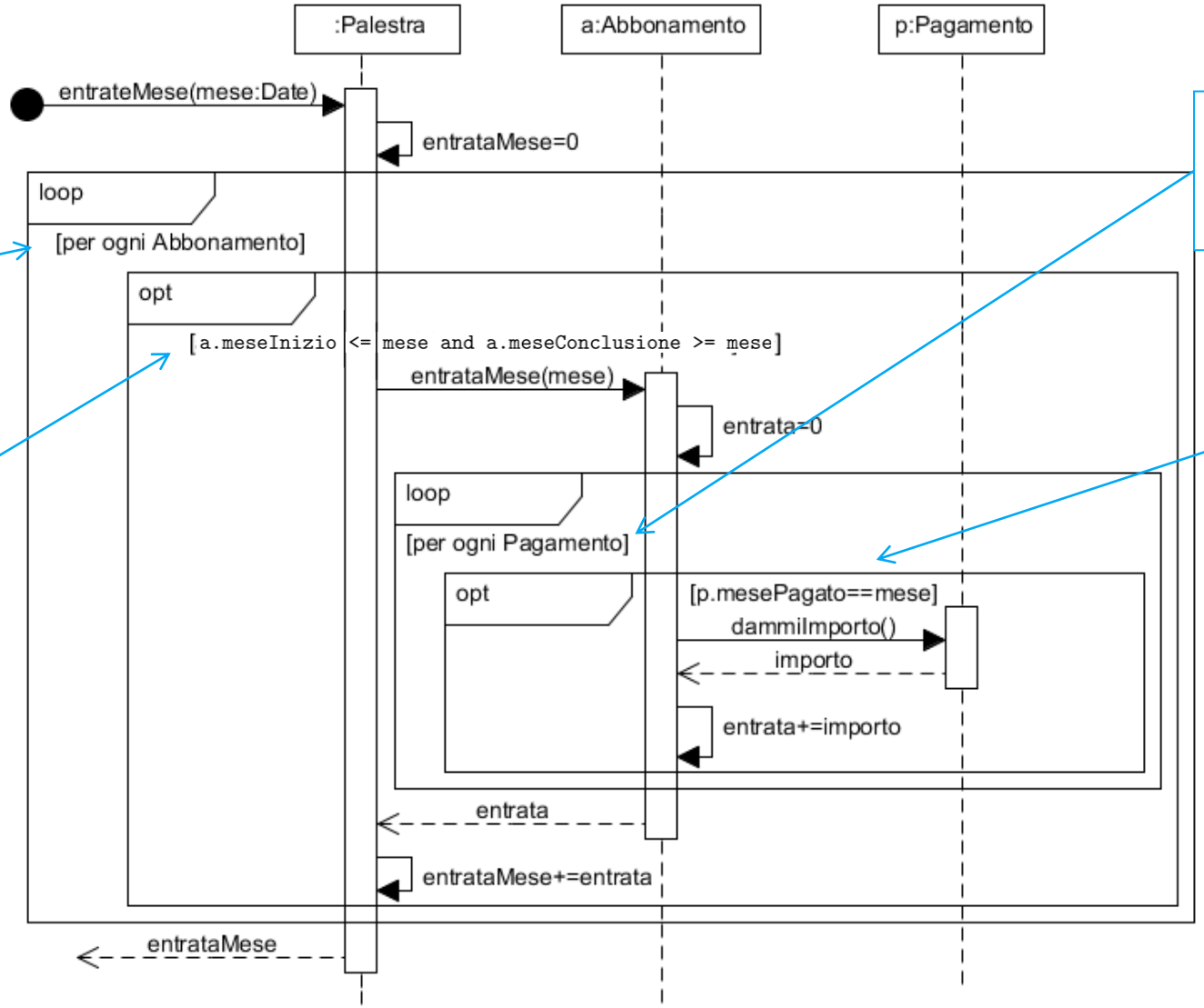
- La classe Palestra rappresenta MegaGym ed è associata ai servizi, agli esercizi e agli abbonamenti disponibili
- questa classe deve mettere a disposizione le operazioni/funzionalità richieste per cui i metodi da progettare dovranno essere definiti in tale ottica
- ogni Abbonamento è associato ai servizi richiesti dall'utente della palestra e al relativo Cliente
- il modello si potrebbe affinare prevedendo una classe Pagamento (vedi prossima slide) per problematiche di contabilità
- la classe Cliente è legata agli eventuali certificati medici, alla propria scheda degli esercizi e al numero di entrate
- la classe associativa **SchedaEsercizi** permette di associare ad uno specifico cliente i propri esercizi che sono offerti dalla palestra (e quindi comuni a tutta la clientela).

# Diagramma concettuale delle classi




- In che modo si potrebbe ottimizzare la gestione dei clienti: aggiungendo quale associazione?
- Quali conseguenze si avrebbero?

# Sequence Diagram: bilancio mensile della palestra



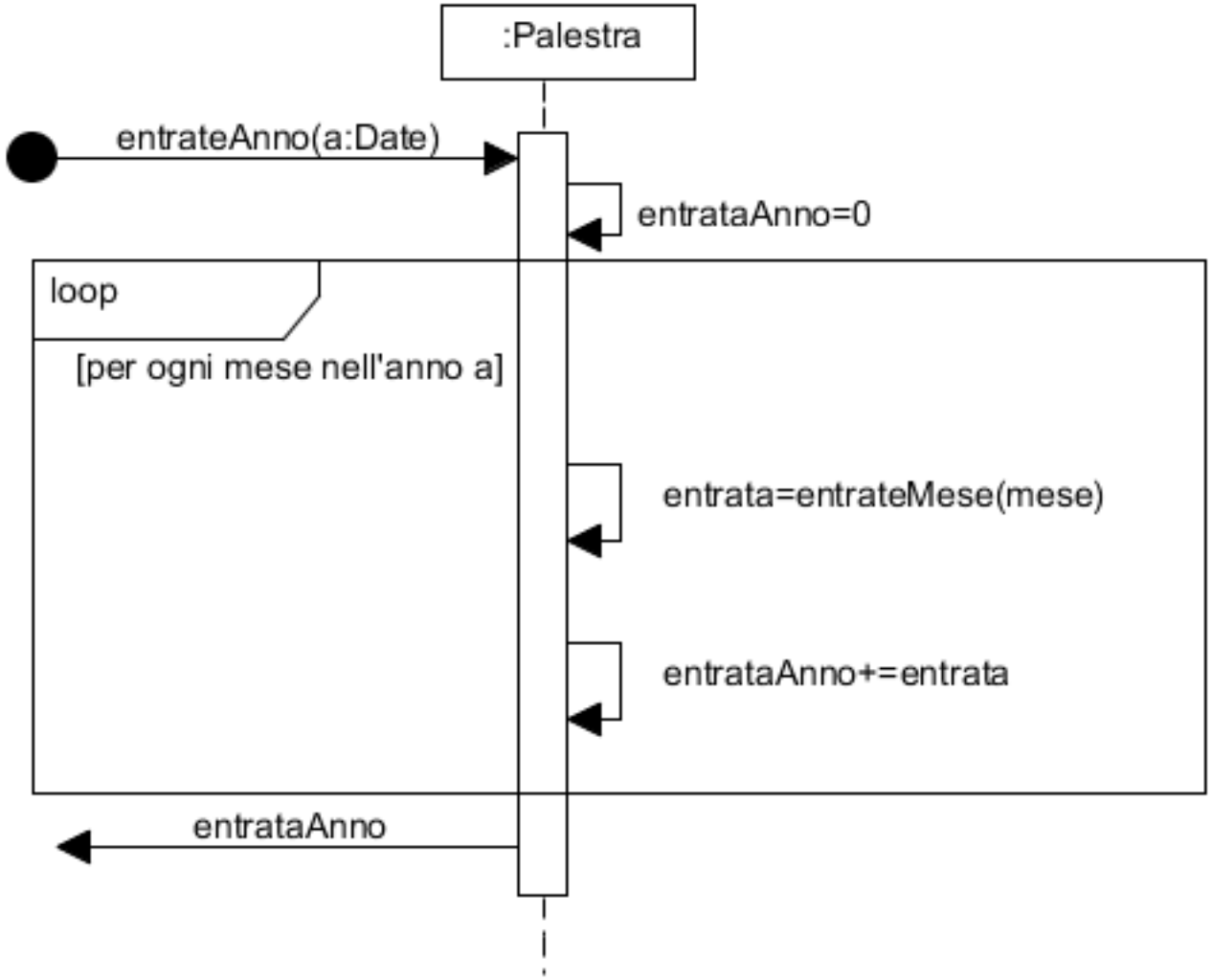

Considero tutti gli abbonamenti dei clienti

Filtro gli abbonamenti utilizzando il mese di interesse: in questo modo verifico che l'abbonamento è valido per quel mese

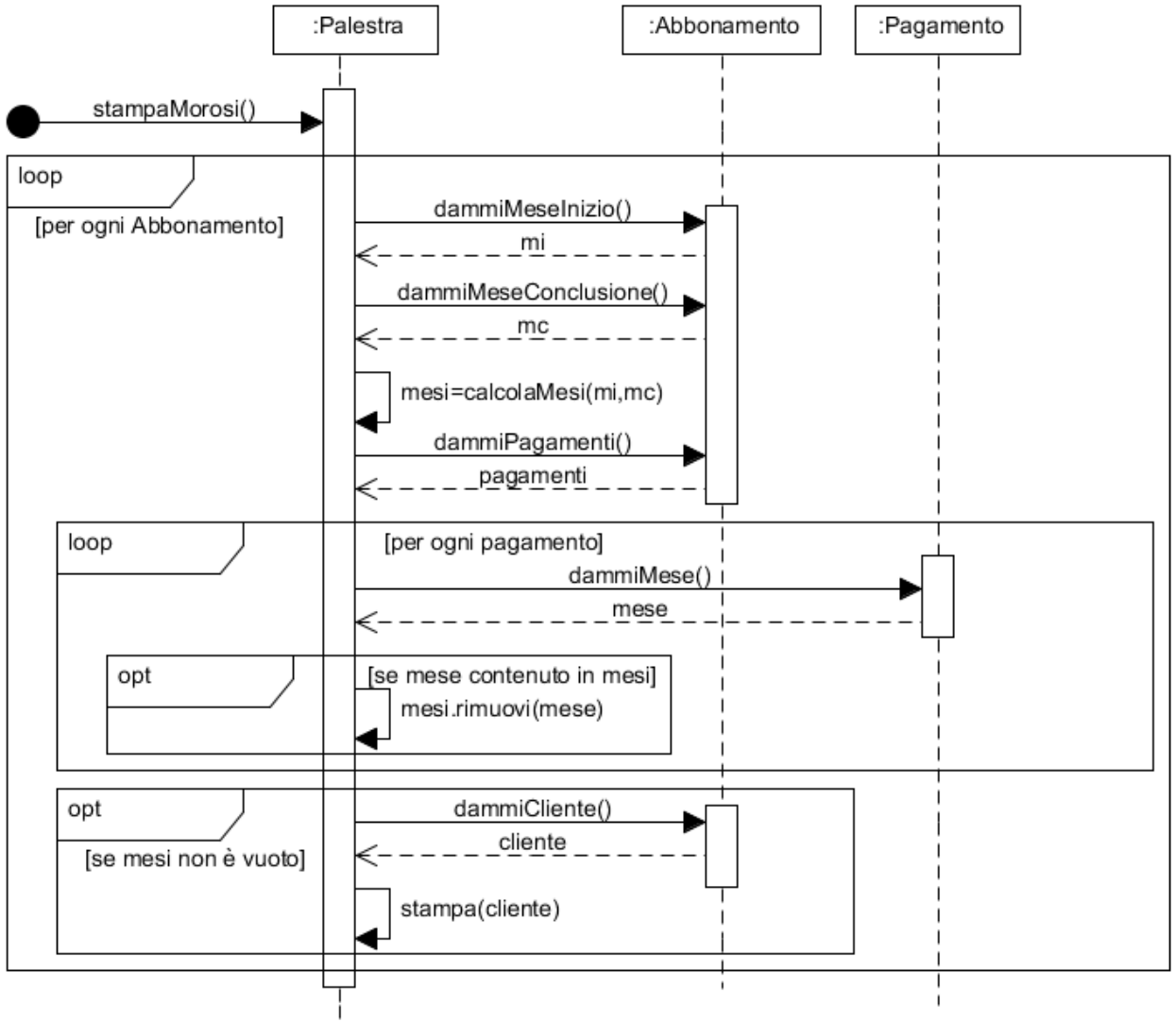
Dell'abbonamento selezionato devo considerare i possibili pagamenti ad esso associato

Filtro sui pagamenti del mese di interesse

# Sequence Diagram: bilancio annuale

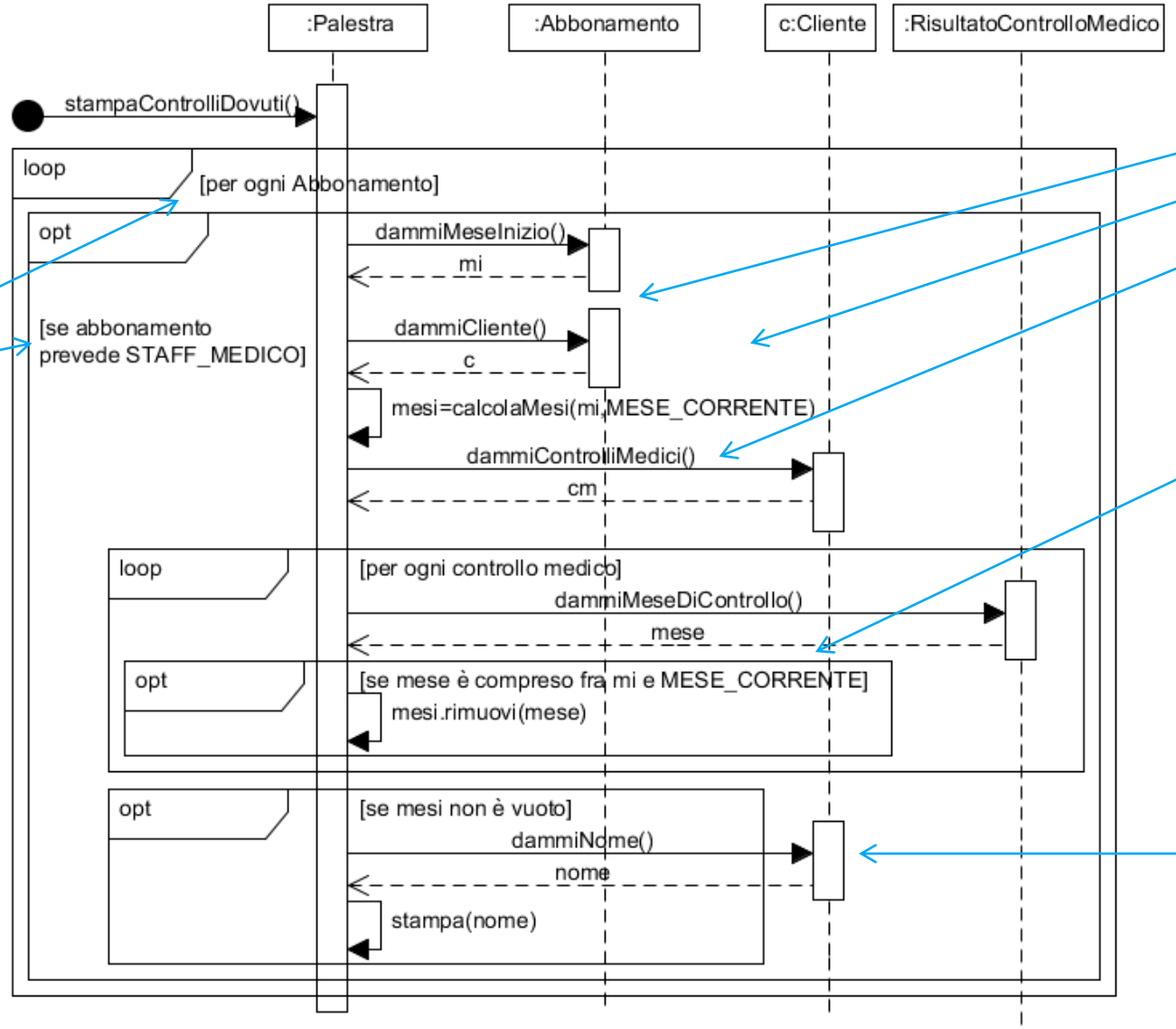


# Sequence Diagram: stampa morosi




- Per ogni abbonamento calcolo i mesi di validità e li pongo nella variabile (presumibilmente un set) *mesi*
- dello specifico abbonamento considero i pagamenti a esso associato
- per ogni pagamento controllo il mese a cui si riferisce ed elimino quest'ultimo da *mesi*
- alla fine di queste operazioni, *mesi* conterrà i mesi non coperti da pagamento
- se *mesi* risulterà «non vuoto» vuol dire che il cliente associato all'abbonamento è moroso

# Sequence Diagram: stampa solleciti per clienti che hanno saltato qualche controllo medico




- considero tutti gli abbonamenti
- filtro considerando i soli abbonamenti per cui è prevista la consulenza medica

Calcolo
- durata mesi
- Cliente
- controlli medici svolti per lo specifico cliente

Tolgo da *mesi* i mesi per cui è stato effettuato un controllo

Se mesi non è vuoto, il cliente non ha svolto tutti i controlli

# Ascensore

# Gestione di un ascensore

In un edificio di diversi piani è presente un ascensore che si vuole controllare mediante un sistema informatizzato.

Lo scenario in questione rispecchia il «normale» funzionamento di un ascensore.

# Requisiti informali

- A ogni **piano** sono presenti due **pulsanti** per la chiamata: uno per scendere, uno per salire (fanno eccezione il primo e ultimo piano).
- Ai piani sono presenti:
  - un display che mostra il piano a cui si trova l'ascensore
  - due frecce luminose che mostrano la direzione dell'ascensore in arrivo
- all'interno dell'ascensore si trovano tanti pulsanti quanti sono i piani. La pressione di un pulsante equivale alla richiesta di raggiungere il piano. Più pulsanti premuti costituiscono un insieme di richieste pendenti.
- ogni pulsante è dotato di **lampadina** (la sua accensione indica la richiesta notificata).

# Requisiti informali

- L’ascensore è dotato di **porta** automatica (con **fotocellula**)
- A ogni piano è collocato un **sensore** (passaggio o presenza dell’ascensore)
- Quando non impegnato, l’ascensore rimane fermo all’ultimo piano servito, con la porta aperta

A questi si aggiungono i «normali» requisiti di funzionamento (ad es.*: quando l’ascensore si muove, le porte devono essere chiuse*)

Si adottano le seguenti ipotesi semplificative:

- si trascura la possibilità di malfunzionamenti
- non si richiedono politiche di gestione che ottimizzino i consumi e/o i tempi di attesa.

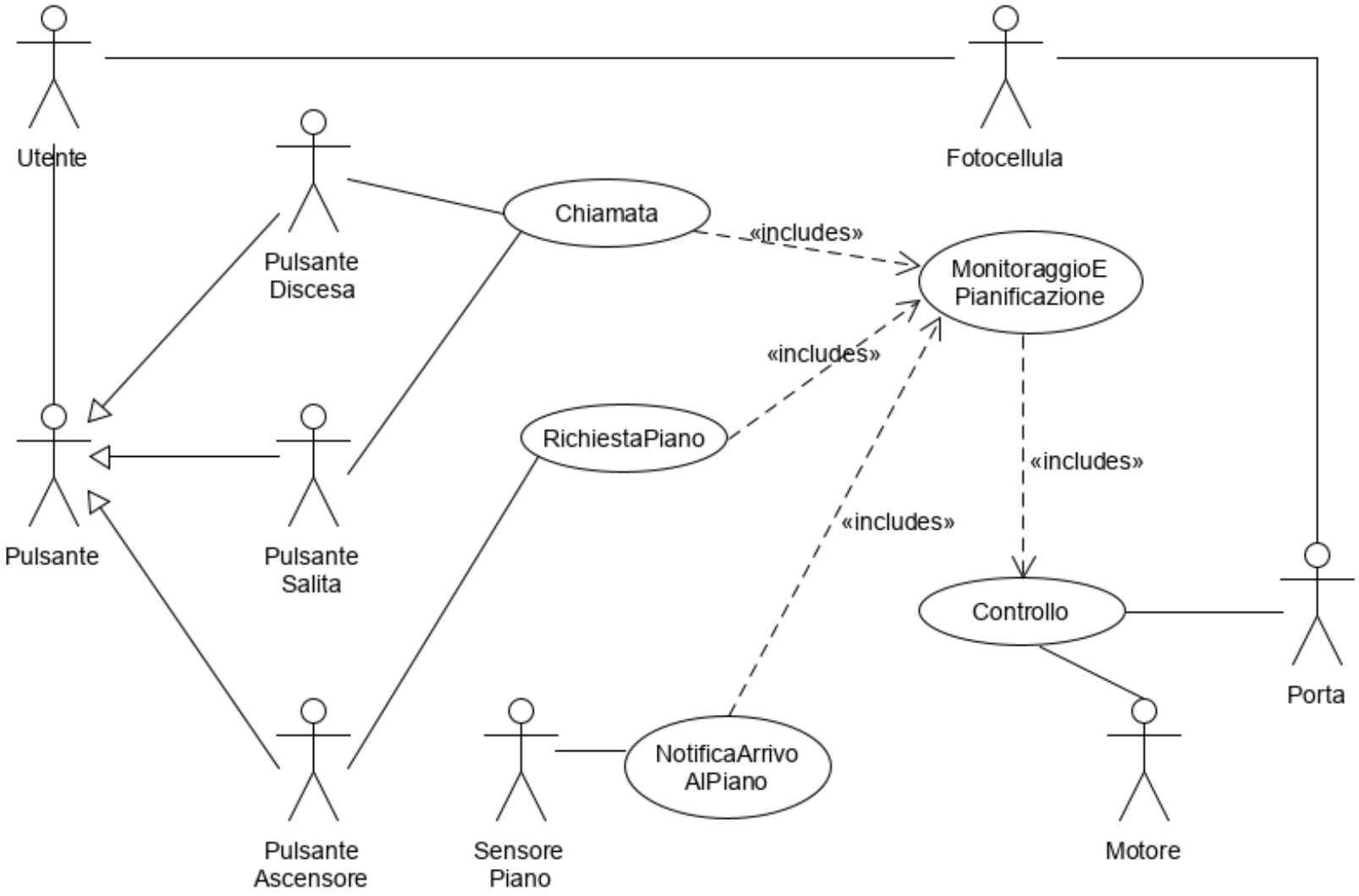
# Analisi del problema

Il problema in esame è un tipico problema di controllo.

Abbiamo cioè un *sistema di cui vogliamo vincolarne il comportamento in relazione agli stimoli provenienti dall'ambiente esterno* in modo da soddisfare le richieste utente.

Il sistema da realizzare è unicamente costituito dal sistema di controllo mentre l'ascensore, sensori, pulsanti, etc. sono elementi dell'ambiente esterno (confini del sistema).

# Casi d’uso


Utente
Fotocellula
Pulsante Discesa
Chiamata
«includes»
MonitoraggioE Pianificazione
«includes»
RichiestaPiano
«includes»
«includes»
Pulsante
Pulsante Salita
Controllo
Porta
NotificaArrivo AlPiano
Pulsante Ascensore
Sensore Piano
Motore

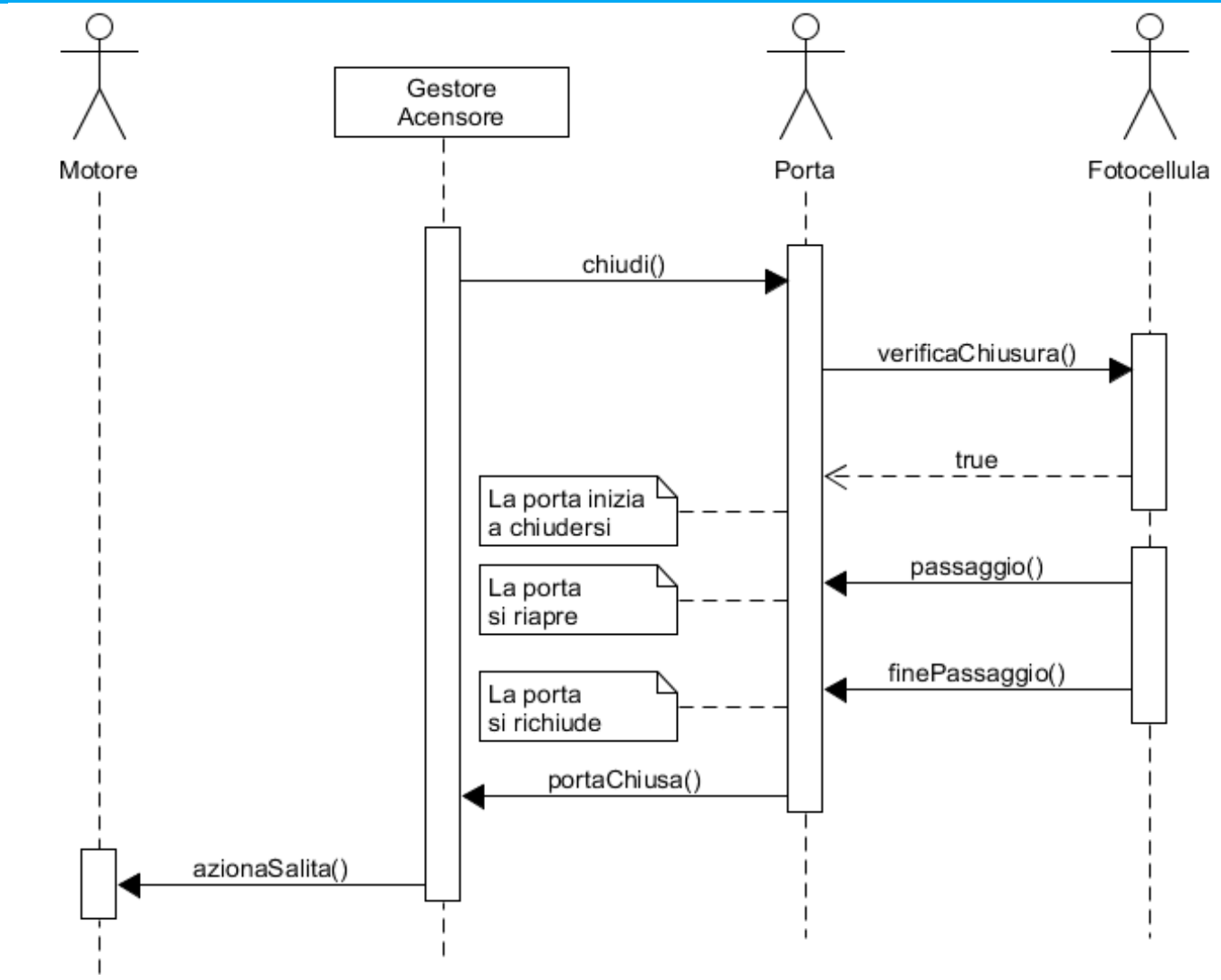
# Casi d’uso

Il sistema è dotato di tre servizi di raccolta degli input dell’ambiente esterno (pulsanti piano + pulsanti interni alla cabina + notifiche raggiungimento di un piano).

Le funzionalità che gestiscono gli input invocano il servizio di monitoraggio e pianificazione (per dedurre la situazione/stato dell’ascensore e determinare l’elenco dei compiti futuri).

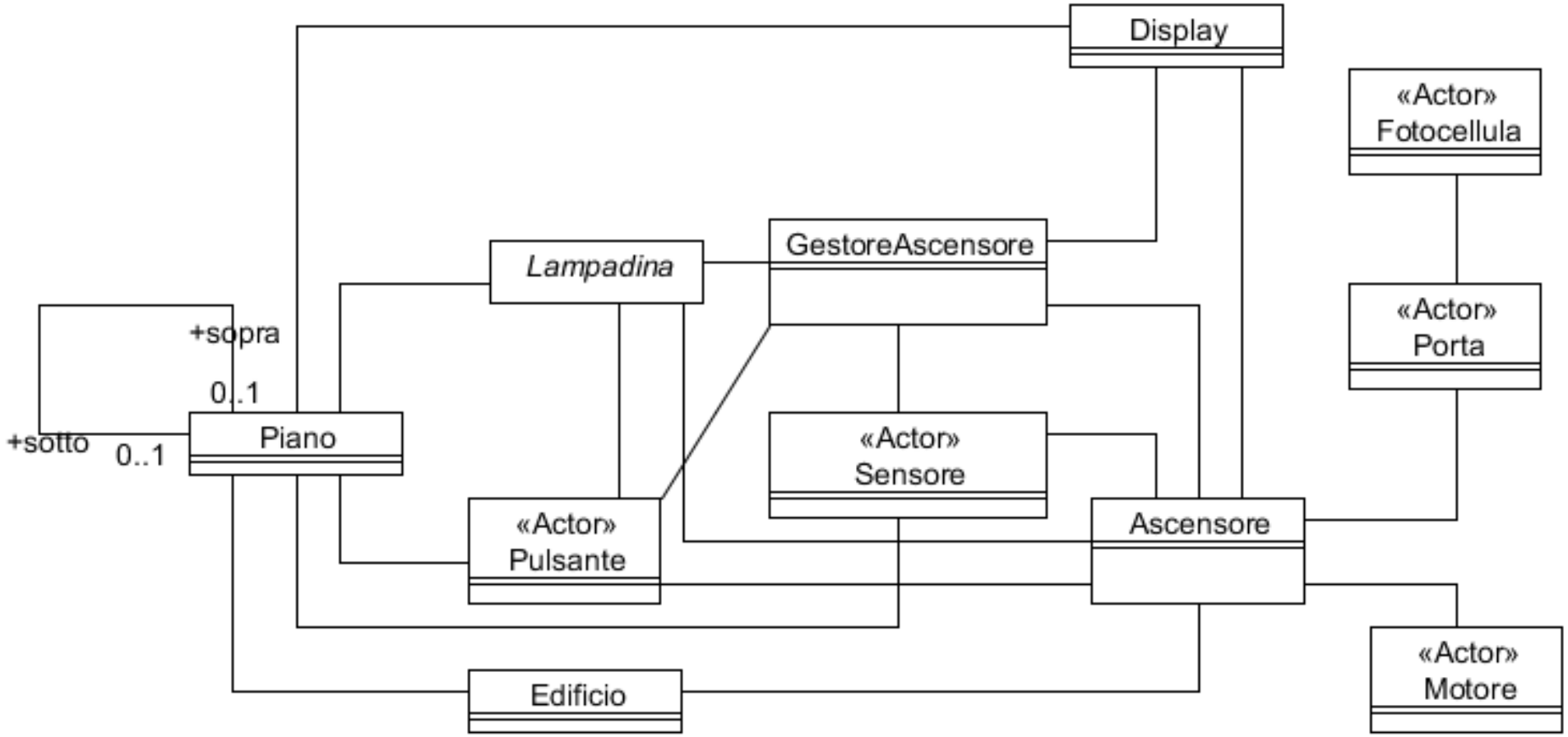
I «compiti» dell’ascensore vengono comunicati al controllore vero e proprio. Quest’ultimo emette i comandi destinati ai componenti dell’ascensore.

# Sequence diagram



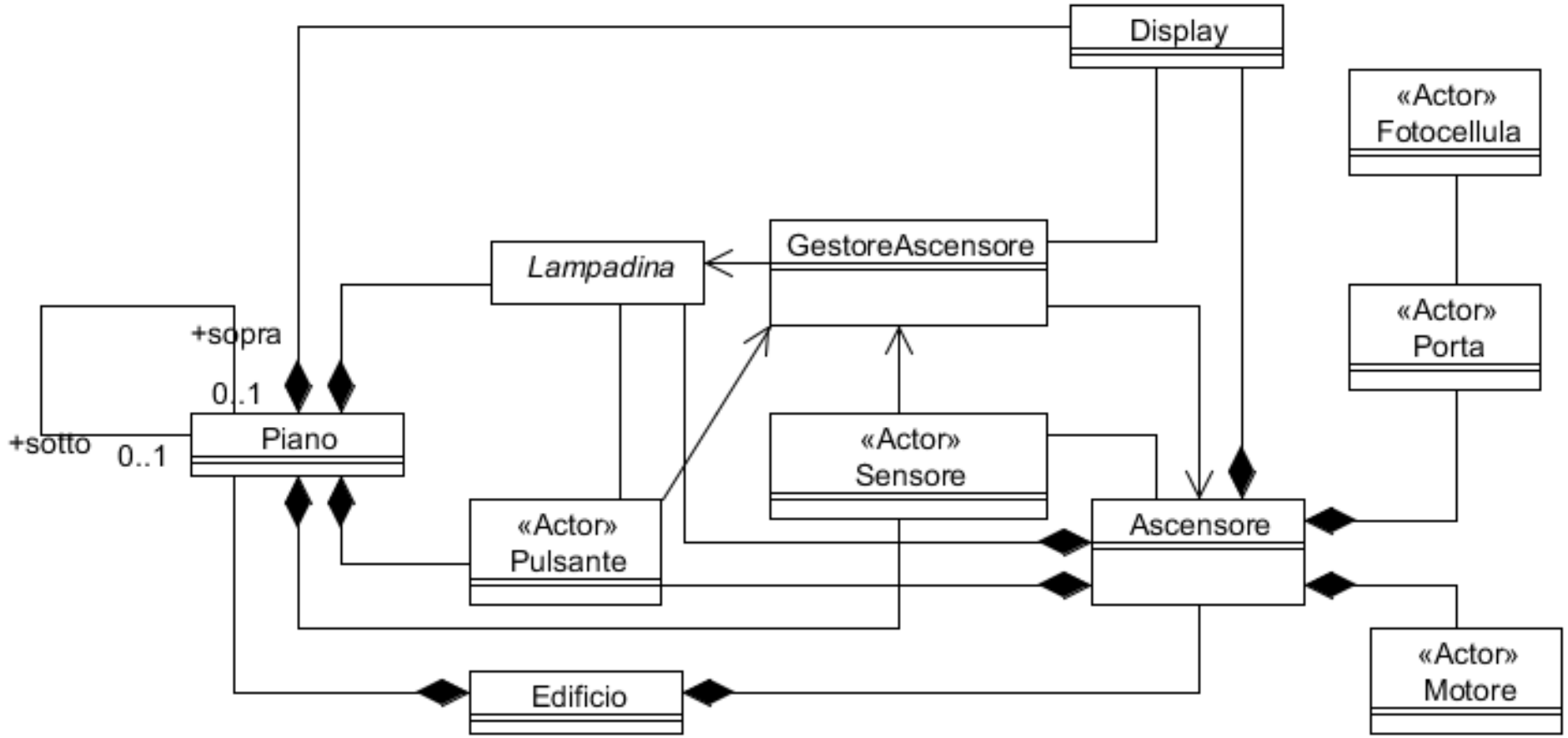

- I diagrammi di sequenza modellano bene solo gli scenari scanditi da eventi.
- Non vi è modo di rappresentare i cambi di stato o vincoli su essi (e.g. le porte sono aperte **se e solo se** l'ascensore è fermo ad un piano).
- Si deve ricorrere a commenti o notazioni non previste dal linguaggio.

# Class diagram (concettuale)



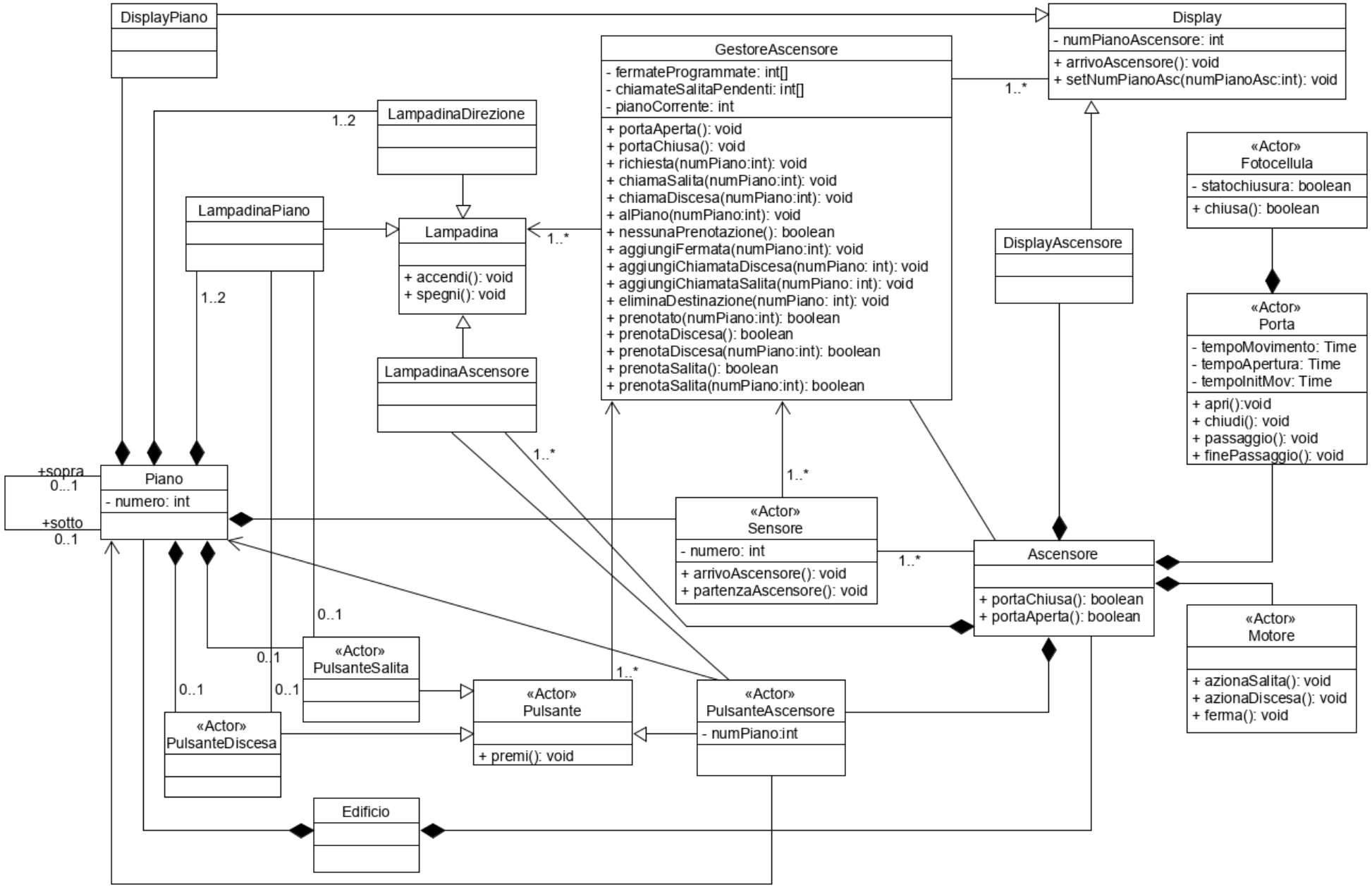

Il piano non è connesso direttamente al gestore dell'ascensore, ma solo attraverso i pulsanti, le lampadine e il sensore.

# Class diagram (concettuale)

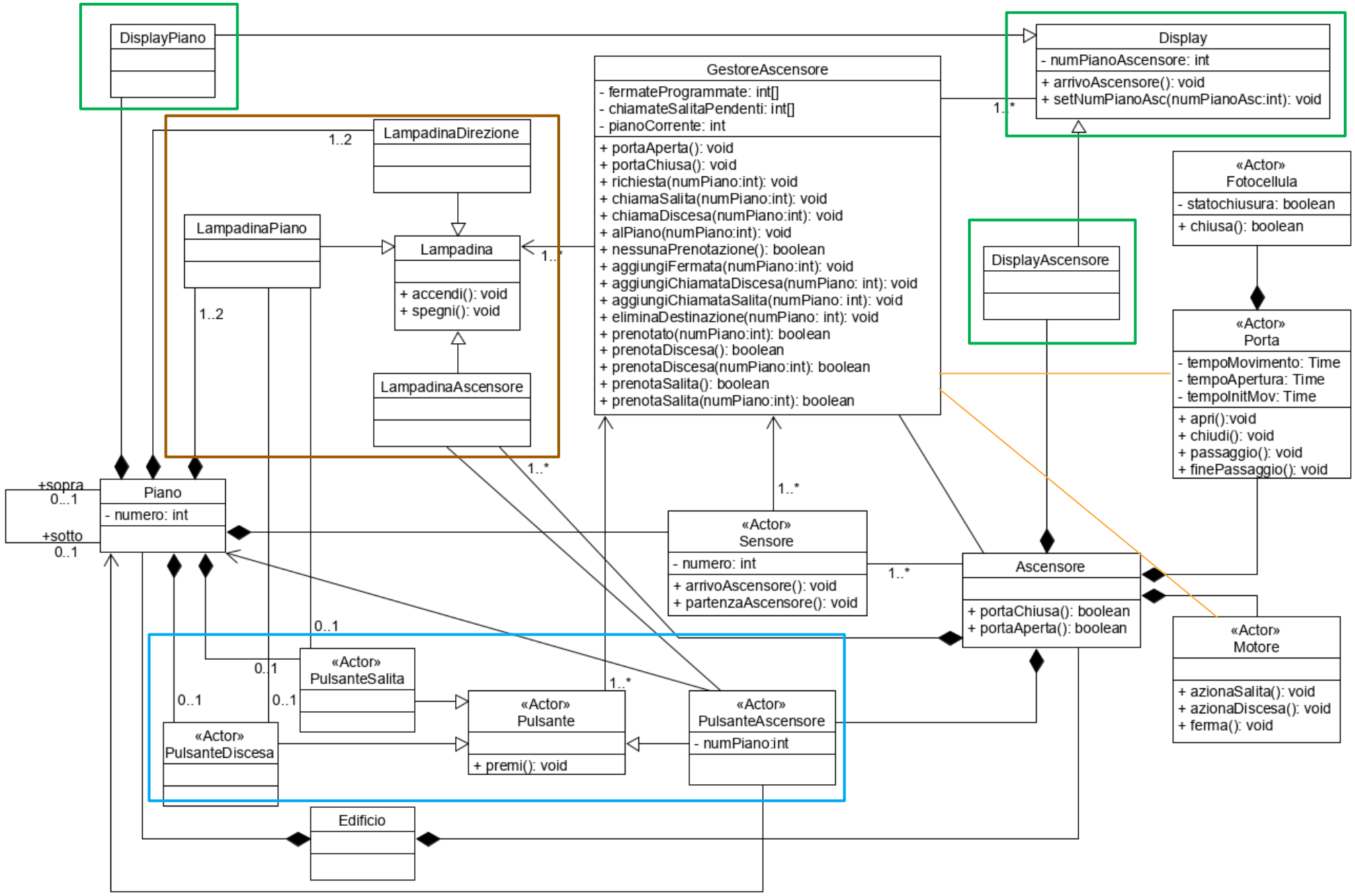



Il flusso delle informazioni/conoscenze deve essere ben preciso: il gestore riceve informazioni da sensori e pulsanti, invia comandi all'ascensore e alle lampadine.

# Class diagram (concettuale)



# Class diagram (concettuale)



# Class diagram (concettuale)

- Nel diagramma sono riportate tre classi (potenzialmente tutte astratte) che vengono poi specializzate: Display, Pulsante, Lampadina.
- Le classi derivate non aggiungono nuovi metodi/funzionalità.
- Quello che contraddistingue realmente le classi derivate è la relazione con l'oggetto servito.
- Nel caso di Display, le classi eredi sono rispettivamente connesse a Piano e Ascensore.
- Essendo invece Pulsante e Display connesse a Gestore, questo implica che tutte le loro sottoclassi sono connesse allo stesso modo.

# Statechart diagram

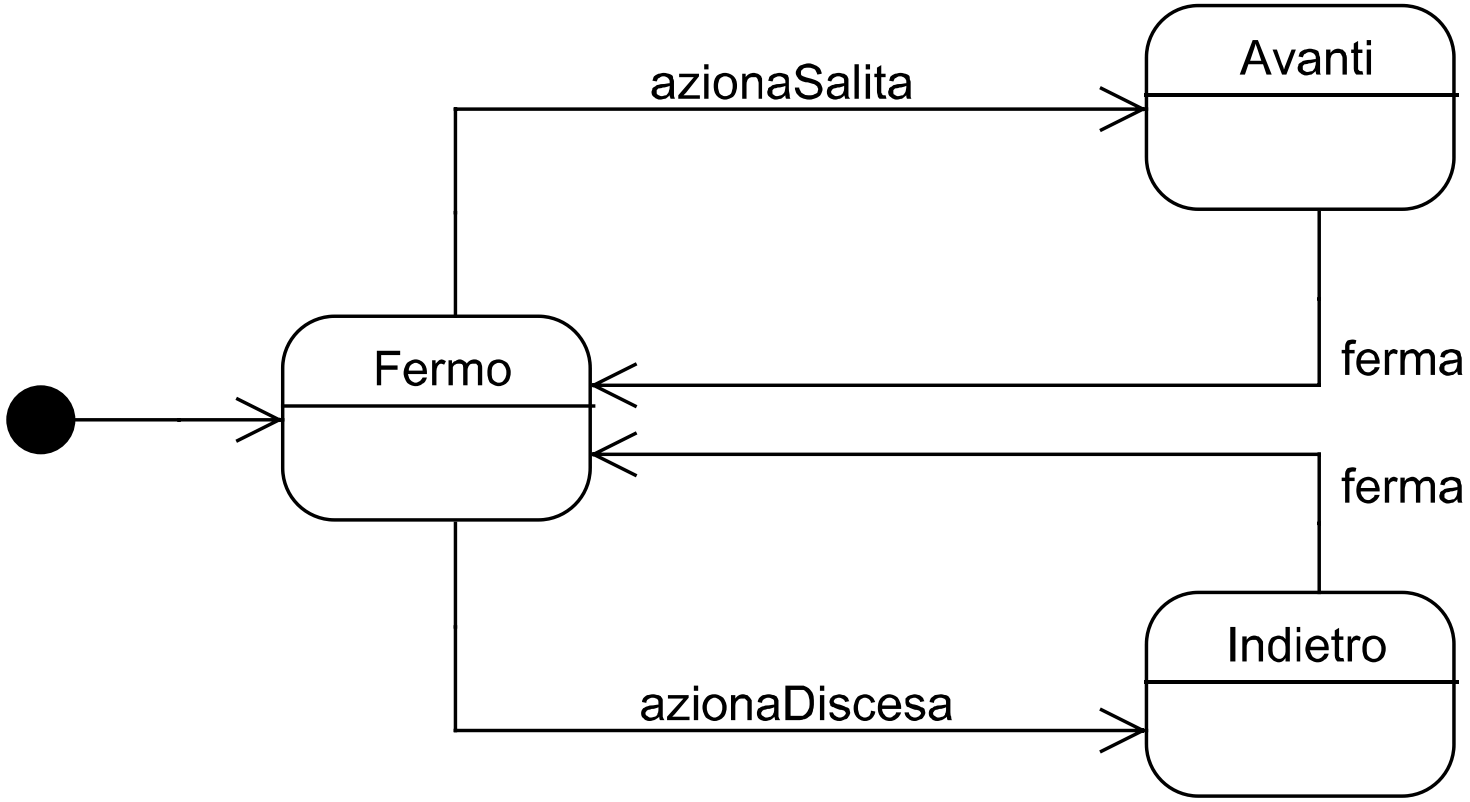
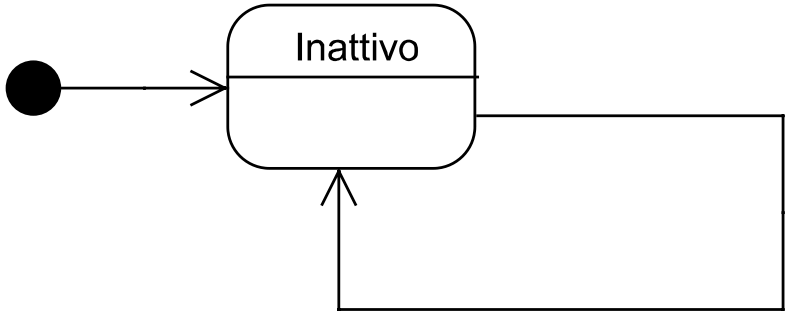



Diagramma degli stati del motore dell'ascensore

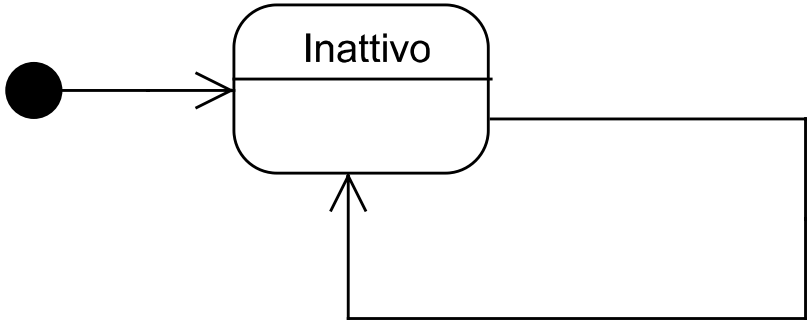
# Statechart diagram




premi/gestoreAscensore.chiamataSalita(piano.numero)

- Comportamento dei pulsanti ai piani per la chiamata a salire (classe PulsanteSalita)
- I pulsanti per scendere hanno comportamento simile

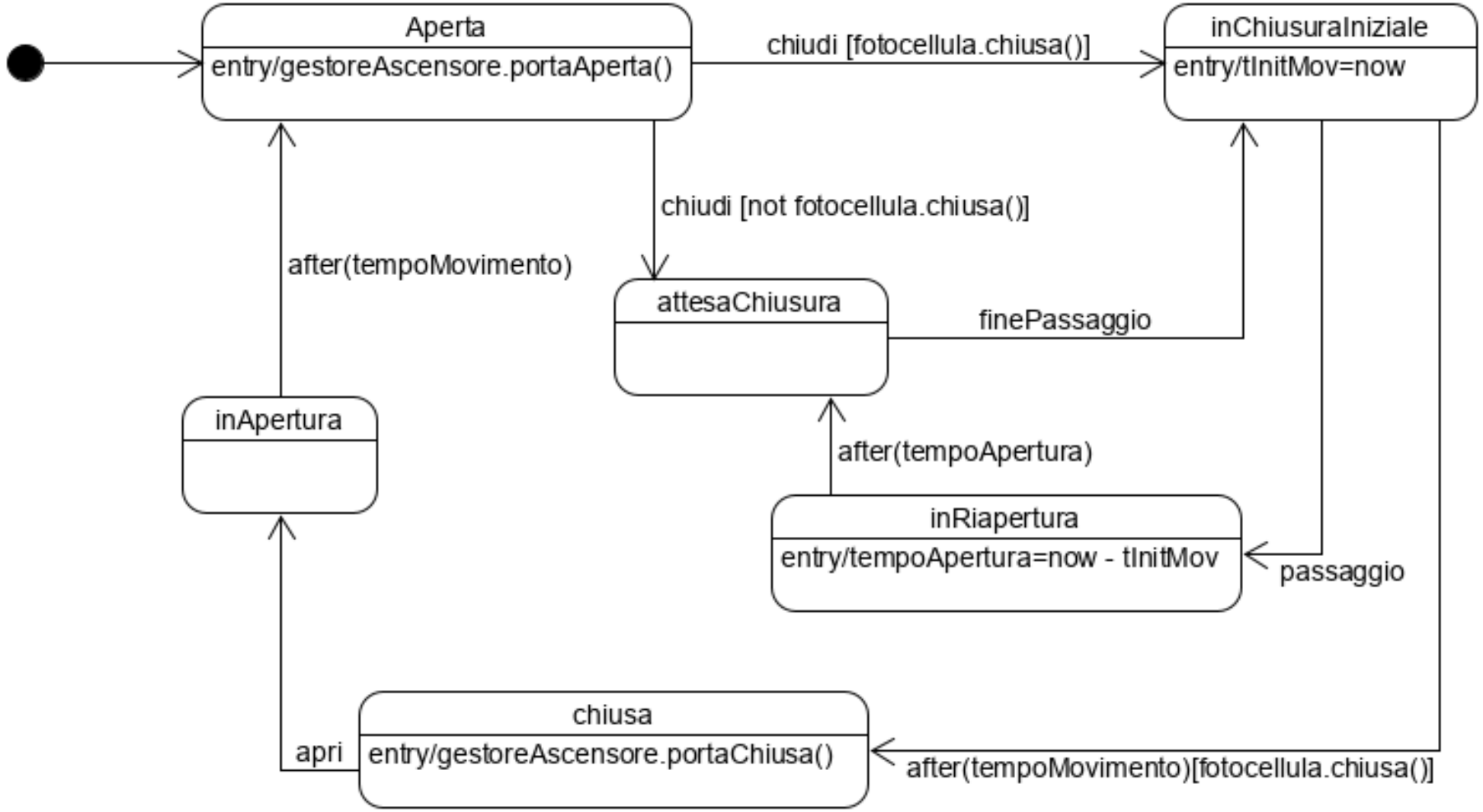
- Comportamento dei pulsanti presenti nell'ascensore (classe PulsanteAscensore)




premi/gestoreAscensore.richiesta(numPiano)

In questo caso (comportamenti non complessi), lo scopo è specificare come reagire agli stimoli esterni e quali messaggi devono essere generati.

# Statechart diagram: porta dell'ascensore (1/3)



# Statechart diagram: porta dell’ascensore (2/3)

- La porta ci mette un certo tempo ad aprirsi e a chiudersi: oltre agli stati *Aperta* e *Chiusa* è stato necessario introdurre degli stati intermedi (come *InApertura*).
- Mentre la porta si sta chiudendo (*InChiusuraIniziale*) è possibile che qualcuno entri o esca: la fotocellula segnala il passaggio e la porta si riapre.
- La riapertura non è istantanea: la porta andrà prima in uno stato *InRiapertura*, poi resterà aperta (*AttesaChiusura*) finché la fotocellula non darà il via libera, dopo di che ricomincerà a chiudersi (stato *InChiusuraIniziale*).

**NOTA: Comportamento della fotocellula: il circuito risulta chiuso quando non c'è alcun passaggio; in corrispondenza dei passaggi il circuito è aperto. L’apertura (passaggio) e chiusura (fine passaggio) sono segnalate esplicitamente mediante messaggi al gestore dell’ascensore**

# Statechart diagram: porta dell’ascensore (3/3)

- L'attributo *chiusa* della fotocellula indica lo stato del circuito, che è chiuso quando non sta passando nessuno e aperto quando la luce della fotocellula è interrotta dal passaggio delle persone attraverso la porta.
- Il tempo di apertura della porta è variabile (questo quando l’apertura è causata dal passaggio di persone), si suppone che la porta ci metta ad aprirsi tanto tempo quanto è stata in chiusura.
- Se qualcuno entra o esce mentre la porta si sta aprendo non c'è alcun effetto sulla porta, che continuerà regolarmente ad aprirsi.

# Statechart diagram: gestore dell'ascensore (1/5)

Chiarimenti e strutture dati di appoggio:

- *fermateProgrammate* rappresenta l'insieme dei piani a cui l'ascensore si fermerà nel suo viaggio corrente
- *chiamateSalitaPendenti* rappresenta l'insieme di chiamate a salire che non possono essere soddisfatte nel viaggio corrente e che quindi lo saranno in seguito (Ad esempio, se l'ascensore sta scendendo e arriva una richiesta di salita, quest'ultima dovrà attendere che l'ascensore abbia concluso la discesa per poter essere soddisfatta)
- *chiamateDiscesaPendenti* rappresenta l'insieme di chiamate a scendere che non possono essere soddisfatte nel viaggio corrente (Ad esempio se l'ascensore è al quarto piano e sta salendo, e arriva una chiamata a salire dal secondo piano, quest'ultima dovrà attendere che l'ascensore abbia concluso la salita e l'eventuale successivo viaggio in discesa)

# Statechart diagram: gestore dell'ascensore (2/5)

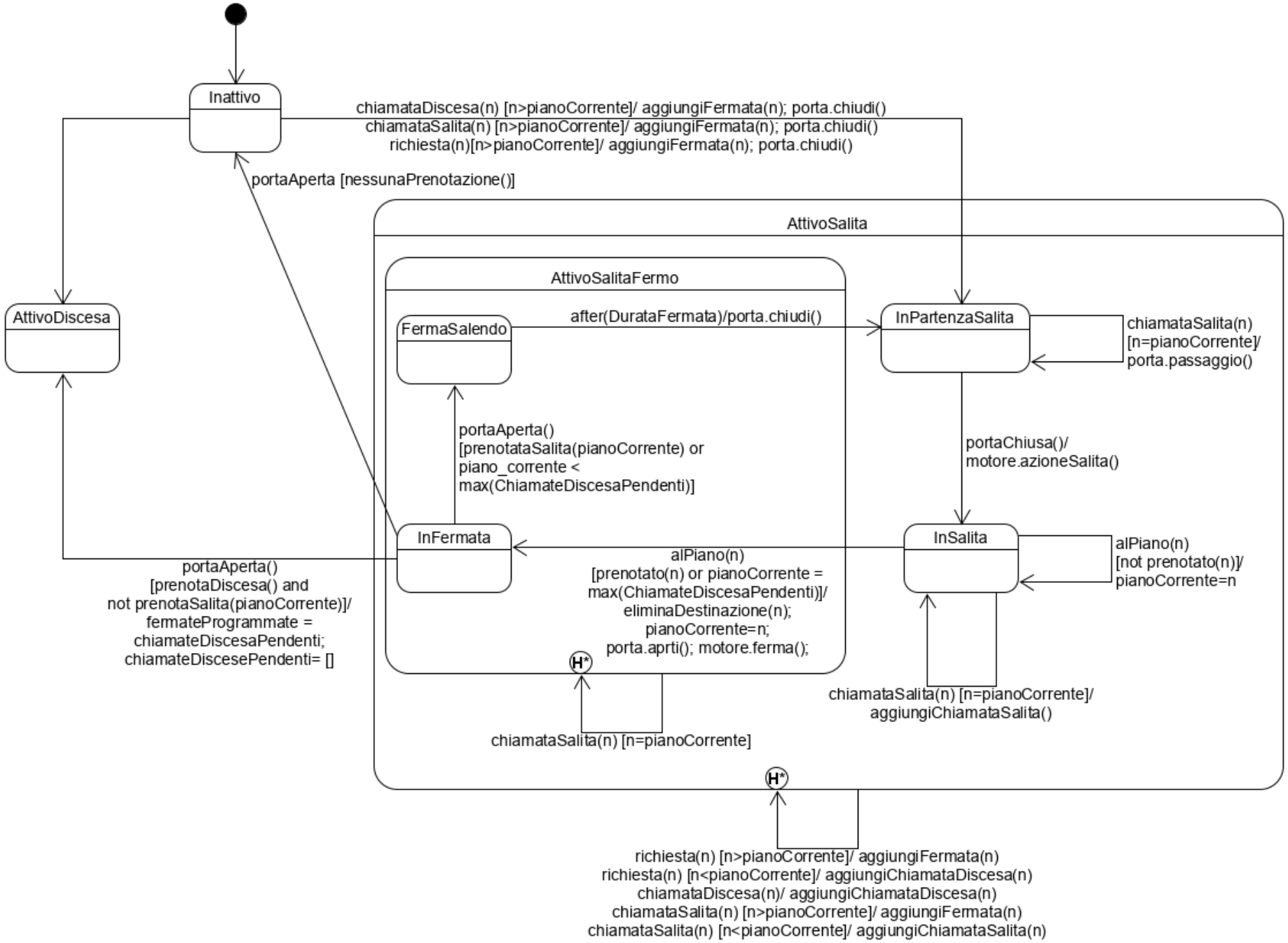
Ipotizzando che l'ascensore stia salendo e si trovi tra il piano X (piano corrente) e il piano X+1, la logica di gestione delle chiamate e delle richieste è la seguente:

- richiesta interna per un piano >= X+1: il piano X+1 viene inserito nel programma di viaggio, cioè in *fermateProgrammate*
- richiesta interna per un piano <= X: viene inserita tra le richieste a scendere pendenti
- chiamata a salire da piano >= X+1: viene inserita nel programma di viaggio
- chiamata a salire da piano < X: viene inserita tra le richieste a salire pendenti.

# Statechart diagram: gestore dell'ascensore (3/5)

- Chiamata a salire da piano = X: dipende dal sottostato
  - se l'ascensore è fermo con le porte aperte possiamo ignorare la richiesta, poiché l'utente può semplicemente entrare e selezionare il piano desiderato
  - se invece l'ascensore è in salita possiamo solo inserirla tra le richieste a salire pendenti
  - se l'ascensore è fermo in attesa che le porte si chiudano si impartisce il comando di apertura porte, in modo da lasciare entrare l'utente che desidera salire
- chiamata a scendere da un piano qualunque: viene inserita tra le richieste a scendere pendenti
- *prenotataSalita(pianoCorrente)* è falsa quando non ci sono ulteriori fermate programmate a salire.

# Statechart diagram: gestore ascensore



# Statechart diagram: gestore dell'ascensore (4/5)

- Per semplificare, è stato indicato solo il comportamento dell'ascensore rispetto al movimento in salita (la parte relativa alla discesa è simmetrica).
- Lo stato *Inattivo* corrisponde all'assenza di richieste e all'ascensore fermo con le porte aperte.
- Quando l'ascensore è inattivo e arriva una chiamata da un piano superiore o una richiesta (generata da un pulsante interno all'ascensore) relativa a un piano superiore, il gestore aggiunge il piano in questione all'insieme delle fermate programmate, impartisce l'ordine di chiusura alla porta e si pone nello stato *InPartenzaSalita.*

# Statechart diagram: gestore dell'ascensore (5/5)

- Nello stato *InPartenzaSalita* si attende la chiusura della porta. Quando questo evento ha luogo, si avvia il motore e la salita inizia (stato *InSalita*).
- Nello stato *InSalita* si verifica prima o poi il raggiungimento di un piano, evento segnalato da *alPiano(NumPiano)*. Quando questo evento ha luogo, si deve aggiornare il piano corrente. Inoltre, se il piano appartiene all'insieme di quelli per cui è pendente una prenotazione, occorre fermarsi.
- la fermata a un piano prenotato avviene attraverso lo stato *InFermata*, in cui si attende l'apertura delle porte.