

La notazione UML

Analisi dei requisiti: obiettivi

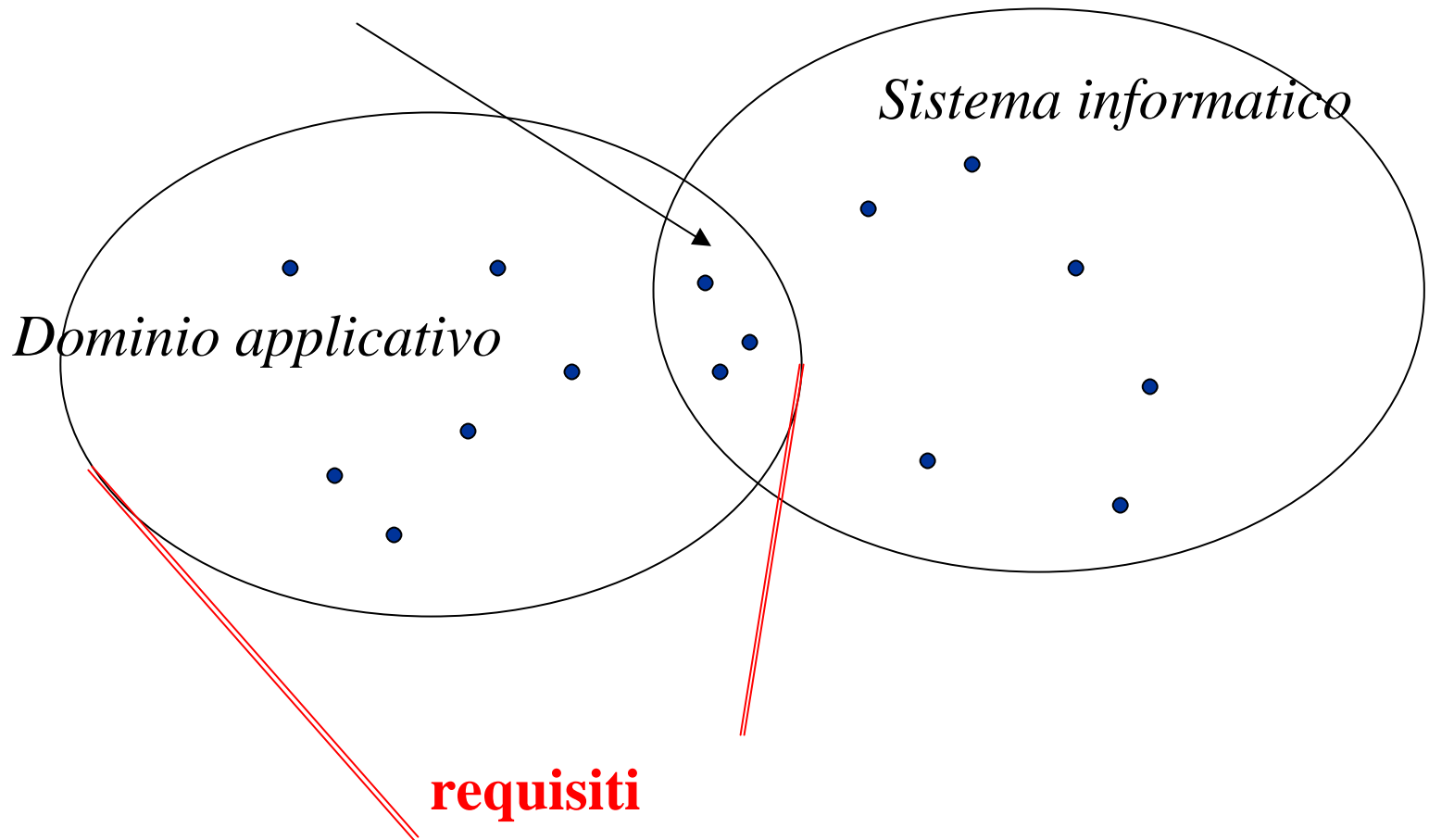
- ✓ Definire requisiti funzionali e non funzionali (come prestazioni, affidabilità, ...) del sistema
 - Descrivere non solo il sistema da realizzare ma anche *l'ambiente* in cui esso dovrà operare
 - Capire il dominio del problema e le responsabilità del sistema da realizzare
 - Comunicazione tra cliente e sviluppatori, e tra sviluppatori
 - Prefigurare e gestire le frequenti modifiche dei requisiti
 - Riutilizzo di conoscenza pregressa del dominio applicativo

Modello di un sistema

- ✓ Un sistema software può essere modellato a diversi livelli e per diversi scopi
- ✓ **Modello esterno:** utile in fase di definizione dei requisiti
 - descrizione astratta del sistema di cui si stanno analizzando i requisiti, in cui tipicamente è inclusa anche una descrizione dell'ambiente in cui il sistema dovrà operare
 - usato per comunicare con i committenti
 - per comunicare ai progettisti che cosa devono realizzare
 - fornisce una descrizione di come il sistema da progettare interagisce con il mondo esterno
 - Potrebbe essere "animato" e analizzato (simulazione) e quindi fungere da prototipo
- ✓ **Modello interno:** necessario in fase di realizzazione e manutenzione
 - Documenta come il sistema è fatto

Dominio applicativo e Sistema Informatico

Dominio condiviso



Modello di un sistema

- ✓ Diversi modelli presentano il sistema da diverse prospettive
 - Esterna: Mostrare il contesto o l'ambiente
 - Comportamentale (Behavioural): operazione svolte dal sistema
 - Strutturale: architettura del sistema
- ✓ Modello può venire
 - in aiuto gli analisti per capire le funzionalità del sistema
 - usato per comunicare con i committenti e i progettisti
 - animato (simulazione) e quindi fungere da prototipo
 - analizzato con valutazioni qualitative, quantitative, studiandone le proprietà`
- ✓ Generazione e analisi degli scenari facilitata se si dispone di un *modello* del sistema e dell'ambiente:
 - descrizione astratta del sistema di cui si stanno analizzando i requisiti, in cui è inclusa anche una descrizione *dell'ambiente* in cui il sistema dovrà operare

UML



- ✓ UML: Unified Modeling Language
- ✓ Notazione object-oriented utilizzabile durante le fasi di analisi e design di un sistema.
- ✓ E' diventato uno standard di fatto per object-oriented modelling
 - Evoluzione e unificazione di tre notazioni: Booch (dell'omonimo autore), OMT (di Rumbaugh) e OOSE (di Jacobson).
 - indipendente da qualsiasi linguaggio di programmazione
 - utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni

Diagrammi UML

- ✓ Viste statiche
 - Use Case Diagram
 - Class Diagram
 - Component Diagram
 - Deployment Diagram
- ✓ Viste Dinamiche
 - State Diagram
 - Activity Diagram
 - Interaction Diagram
 - Sequence Diagram
 - Collaboration Diagram

Registrazione Corsi

- ✓ All'inizio dell'anno accademico, l'ufficio amministrativo dell'università fornisce agli studenti la lista di corsi disponibili, attraverso un sistema di registrazione.
- ✓ Il sistema deve consentire agli studenti di selezionare quattro corsi fra quelli disponibili. In aggiunta ogni studente deve indicare due alternative.
- ✓ Nessun corso può avere più di 10 studenti. Corsi con meno di 3 studenti vengono cancellati automaticamente.

Registrazione Corsi (cont.)

- ✓ I professori devono poter accedere al sistema per indicare i corsi in cui vorrebbero insegnare e per vedere gli studenti iscritti ai loro corsi.
- ✓ Il processo di registrazione dura 3 giorni.
 - Il primo giorno è riservato agli studenti del primo anno.
 - Il secondo giorno è riservato agli altri studenti.
 - Durante il terzo giorno si risolvono eventuali conflitti.
- ✓ Finito il processo di registrazione, le informazioni relative ad un studente sono inviate all'ufficio contabile per determinare le tasse da pagare.

Come rappresentare questi requisiti con un modello?

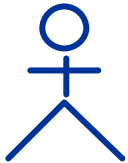
- ✓ Il modello piu` astratto, utile per fissare i requisiti, definisce
 - gli attori
 - i possibili *usi* loro concessi

Use Case Diagram

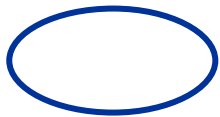
Use Case Diagrams

- ✓ Definiscono il comportamento del sistema
 - Le funzionalità (processi) principali
 - Come il sistema agisce e reagisce
- ✓ Descrivono
 - Il sistema
 - L'ambiente
 - Le relazioni fra sistema e ambiente
- ✓ Diversi livelli di dettaglio
 - Scomposizione gerarchica

Elementi Fondamentali



- ✓ Actor: è qualcuno (utente) o qualcosa (sistemi esterni - hardware) che:
 - Scambia informazioni con il sistema
 - Fornisce input o riceve output dal sistema



- ✓ Use Case: è un'unità funzionale parte del sistema

Relazioni Fondamentali

- ✓ **interazione** indica relazioni tra attori e casi
- ✓ **uses** indica l'uso di una certa funzionalità (utile quando funzionalità simili sono presenti in diversi contesti)
- ✓ **extends** definisce una funzionalità per estensione di una funzionalità già esistente



Use Case

- ✓ Gli use case possono essere ricavati dalle interviste con gli utenti. Si identificano
 - Gli obiettivi: ciò che il sistema dovrebbe fare secondo gli utenti
 - Le interazioni: cosa vorrebbero (potrebbero) fare i diversi utenti
- ✓ Gli use case di alto livello sono volutamente generici
 - I dettagli vanno aggiunti raffinando le funzionalità del sistema

Use Case (cont.)

- ✓ Il processo di definizione degli use case è iterativo
 - Si inizia identificando il comportamento più semplici
 - Si descrivono i comportamenti alternativi e più complessi
- ✓ Quando smettere?
 - Un buon livello di dettaglio facilita tutte le attività successive
 - Troppi dettagli
 - Complicherebbero inutilmente la descrizione
 - Introdurrebbero prematuramente scelte progettuali
 - Precluderebbero la visione d'insieme

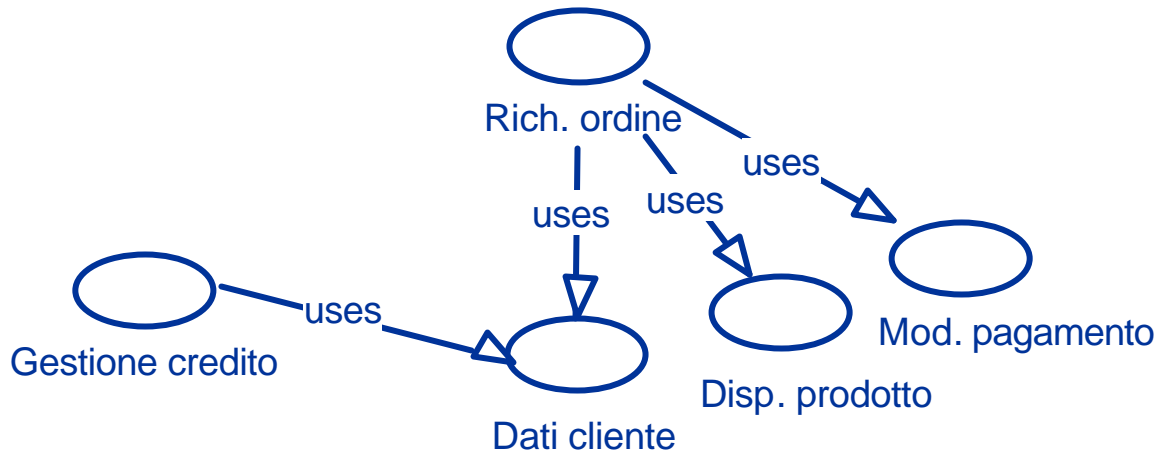
Extends

- ✓ **Extends** indica che uno use case è simile a un altro ma è più specifico.
- ✓ Si descrive lo use case più specifico per differenze rispetto a quello più generale
- ✓ Esempi
 - Pagamento prodotto e pagamento con carta di credito
 - Ordine di un prodotto e ordine via internet:



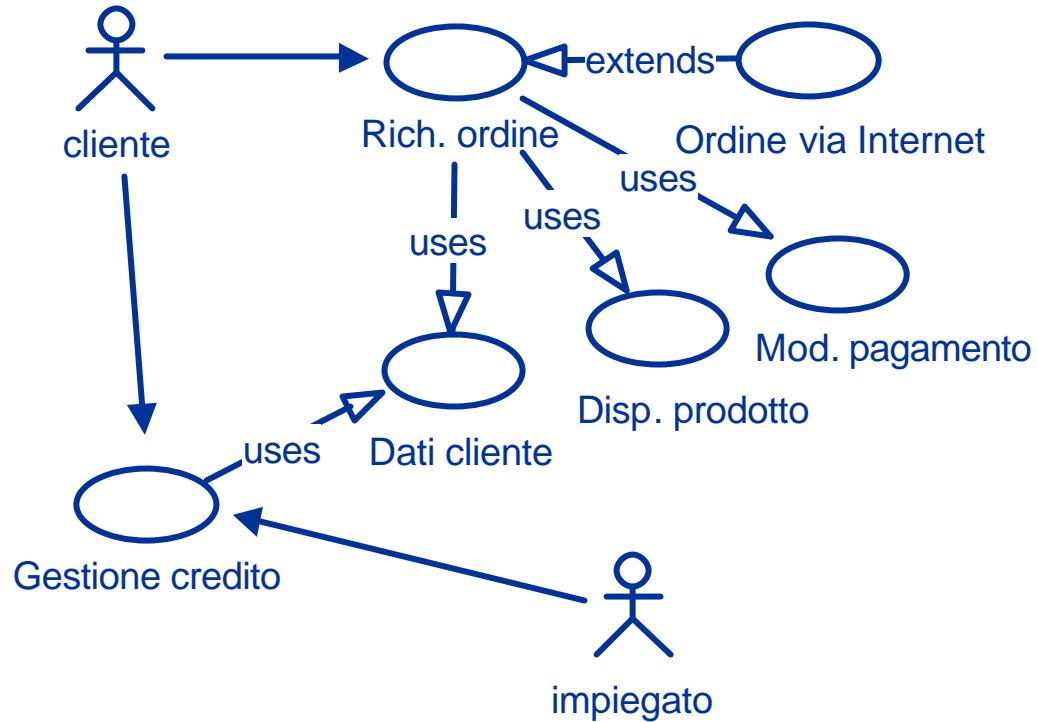
Uses

- ✓ **Uses** fattorizza attività ricorrenti e condivise



- ✓ Analogie con
 - Chiamata a sottoprocedura in un linguaggio di programmazione
 - Scomposizione gerarchica (tramite "catene" di uses...)

Un esempio



Extends vs. Uses

✓ Extends

- Descrive una specializzazione rispetto al caso normale
- Facilita la fattorizzazione (riuso) di comportamenti "simili"

✓ Uses

- Identifica un componente (parte) di un caso più complesso
- Massimizza ed evidenzia i componenti riutilizzabili

Scenari

- ✓ Uno scenario è una realizzazione di uno use case
 - È una "esecuzione" particolare dello use case
 - Rappresenta il comportamento (le azioni e gli eventi) del sistema nel caso particolare considerato
- ✓ Ogni use case dovrebbe essere corredato da un insieme di scenari
 - Scenari principali (più possibile)
 - Tutto funziona correttamente
 - Scenari secondari (pochi e significativi)
 - Eccezioni (eventuali problemi o malfunzionamenti)

Scenari

- ✓ Gli scenari definiscono requisiti di più basso livello rispetto agli use case
- ✓ Gli scenari sono solitamente definiti in linguaggio naturale
- ✓ UML propone una notazione particolare

Interaction Diagram

(di cui noi vedremo solo Sequence Diagram)

- ✓ Quanti scenari si devono definire?
 - Servono tanti scenari quanti sono quelli necessari per capire il corretto funzionamento del sistema e le eccezioni che si ritengono significative in durante l'analisi

Riassumendo

- ✓ L'identificazione degli use case è la prima cosa da fare
 - rappresentano i requisiti del sistema
 - forniscono una base per la pianificazione ed il controllo del progetto
- ✓ Il processo di definizione è un processo iterativo
 - Dal generale al grado di dettaglio richiesto
- ✓ Gli use case vengono descritti in dettaglio in termini di scenari

Esercizio

Si definisca un semplice Use Case Diagram per modellare il sistema informativo di una società di autonoleggio. Il sistema deve gestire le prenotazioni (sia telefoniche, che via Internet), la richiesta diretta di un'autovettura, la fatturazione, l'invio di solleciti in caso di ritardo nella riconsegna e la gestione del parco macchine

Interaction Diagram

- ✓ Descrivono il comportamento dinamico di un gruppo di oggetti che “interagiscono” per risolvere un problema
- ✓ Sono utilizzati per “formalizzare” gli scenari in termini
 - Entità (oggetti)
 - Messaggi scambiati (metodi)
- ✓ UML propone due diversi tipi di Interaction Diagram
 - Collaboration Diagram
 - Sequence Diagram (noi vediamo questi)
 - Evidenziano la sequenza temporale delle azioni
 - Non mostrano le associazioni tra oggetti

Un esempio

(Registrazione Corsi)

Paperino Paolino sceglie i corsi: informatica, fisica, analisi e disegno

Come corsi alternativi sceglie: fotografia e giornalismo

I corsi scelti sono immessi nel sistema di registrazione

Lo studente viene aggiunto ai primi 3 corsi principali

Il quarto corso non è disponibile

IF la prima alternativa è disponibile **THEN**

Lo studente viene aggiunto al corso

ELSIF la seconda alternativa è disponibile **THEN**

Lo studente viene aggiunto al corso

ELSE notifica allo studente

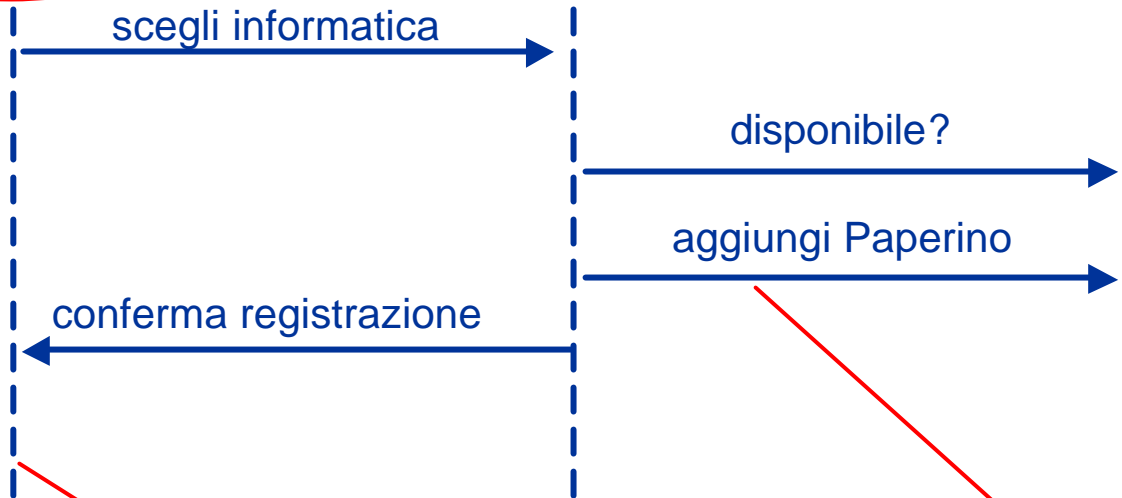
IF not fallisce **THEN**

crea la cartella corsi dello studente

manda le informazioni all'ufficio contabile

Sequence Diagram

oggetti

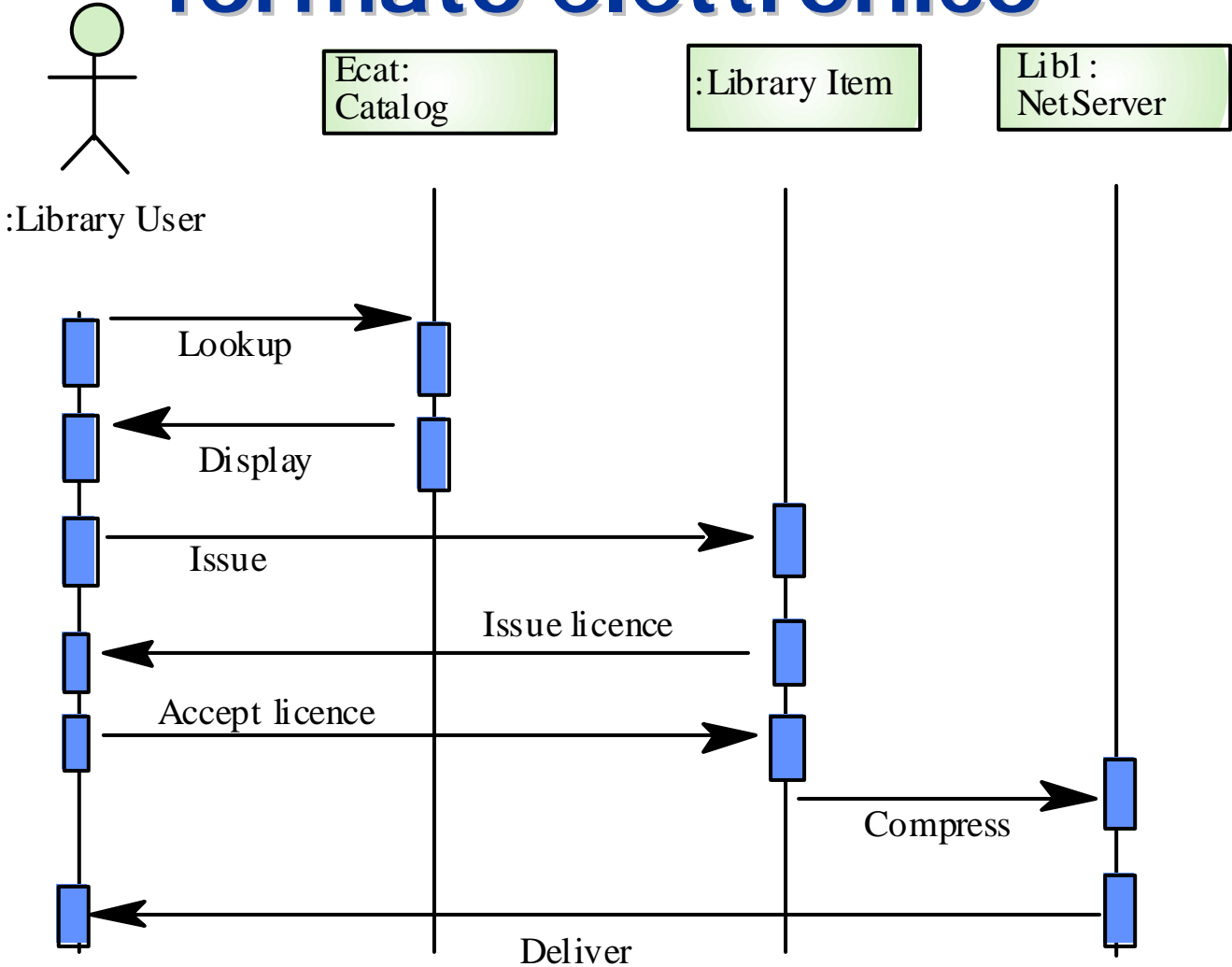


tempo

linea di vita--*lifeline*

messaggio

Distribuzione di materiale in formato elettronico



Esercizio

Si definisca un semplice Sequence Diagram per modellare il prelievo di denaro, supponiamo 200 Euro, da uno sportello Bancomat

Messaggi

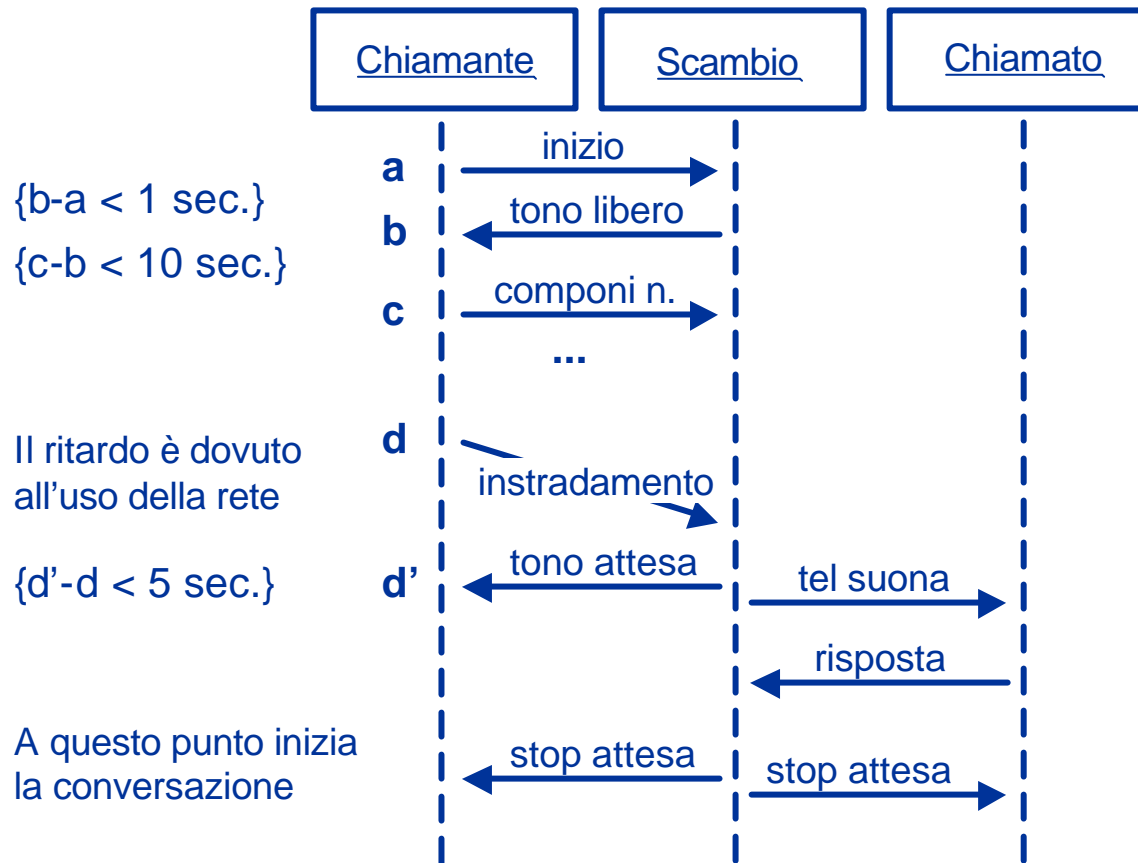
✓ Sincroni

- Chi invia il messaggio aspetta la conclusione dell'operazione richiesta (chiamata di procedura)

✓ Asincroni

- Equivalenti a qualche forma di comunicazione tra processi
- Chi invia il messaggio continua, senza aspettare la conclusione dell'operazione richiesta (comunicazione fra processi)

Sequence Diagram (Sistemi in Tempo Reale)



Riassumendo

- ✓ Uno scenario può essere rappresentato per mezzo di un Interaction Diagram
 - I **Sequence Diagram** mettono più enfasi sulla sequenza delle operazioni e possono indicare l'esistenza dei diversi oggetti
- ✓ I diagrammi vanno completati con commenti testuali per fornire ulteriori dettagli

Diagramma delle classi

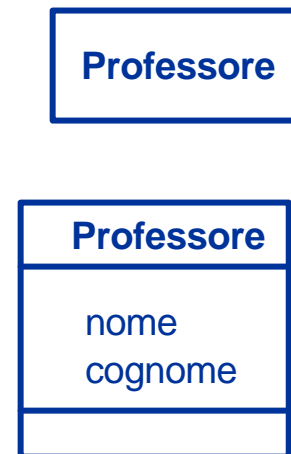
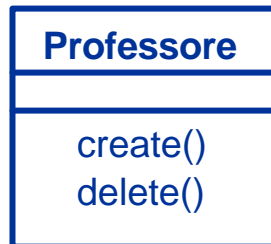
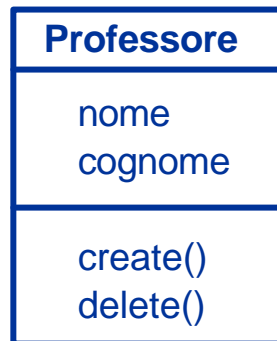
- ✓ Definiscono la visione statica del sistema
 - classi
 - relazioni tra classi
 - associazione (uso)
 - generalizzazione (ereditarietà)
 - aggregazione (contenimento)
- ✓ È forse il modello più importante perché definisce gli elementi base del sistema

Classi durante l'analisi

- ✓ Le classi che si identificano durante la fase di analisi dei requisiti corrispondono alle entità logiche che si trovano nei requisiti e nella descrizione dell'ambiente
- ✓ Non corrispondono alle classi che avremo nell'implementazione

Dettaglio delle classi

- ✓ classe = insieme di entità del sistema o dell'ambiente, con caratteristiche comuni.
- ✓ Classe è composta da tre parti
 - Il nome
 - Gli attributi (lo stato)
 - I metodi (il comportamento)
 - Classe= rettangolo con il nome in cima, attributi in mezzo operazioni in fondo



Le classi del sistema corsi



Trovare le classi e gli oggetti

✓ Dove guardare

- osservazioni di prima mano, ascoltare attivamente, considerare risultati precedenti e altri sistemi, leggere a volontà, prototipi

✓ Cosa cercare

- strutture, altri sistemi, dispositivi, cose ed eventi ricordati, ruoli, procedure operative, luoghi ed unità organizzative

✓ Cosa considerare

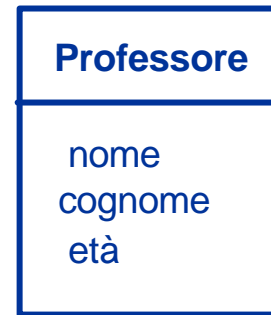
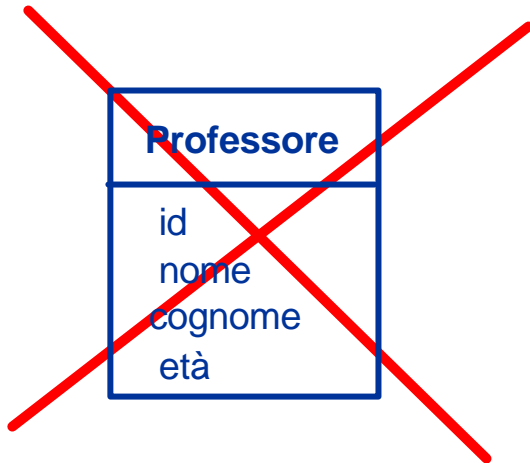
- memoria e comportamento necessari, molteplicità di attributi e di istanze, attributi e metodi sempre presenti, requisiti del dominio

Attributi

- ✓ Un attributo è una caratteristica della classe
- ✓ Gli attributi non hanno identità
- ✓ Ogni attributo deve essere definito in modo preciso
- ✓ *Attributi buoni* per Studente
 - Nome, Cognome, ...
- ✓ *Attributi cattivi*
 - CorsiScelti (i corsi scelti possono essere dedotti a partire dall'elenco degli iscritti di ciascun corso: non serve memorizzarli in attributi)

Attributi (cont.)

- ✓ Gli oggetti avranno una loro identità, non bisogna aggiungerla

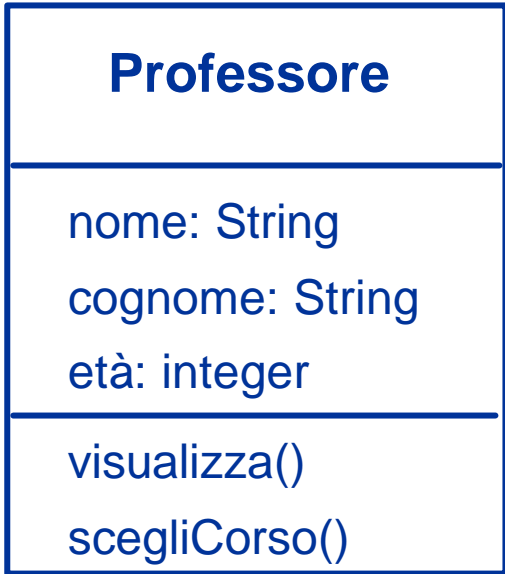


Codice fiscale?

Operazioni (Metodi)

- ✓ Un'operazione è una trasformazione applicabile a un esemplare di una classe
- ✓ Tutte e sole le operazioni rilevanti per il dominio applicativo
 - Bancario
 - ricevePrestito, riceveCredito, apreConto
 - Medico
 - faEsame, prendeMedicina, vaOspedale

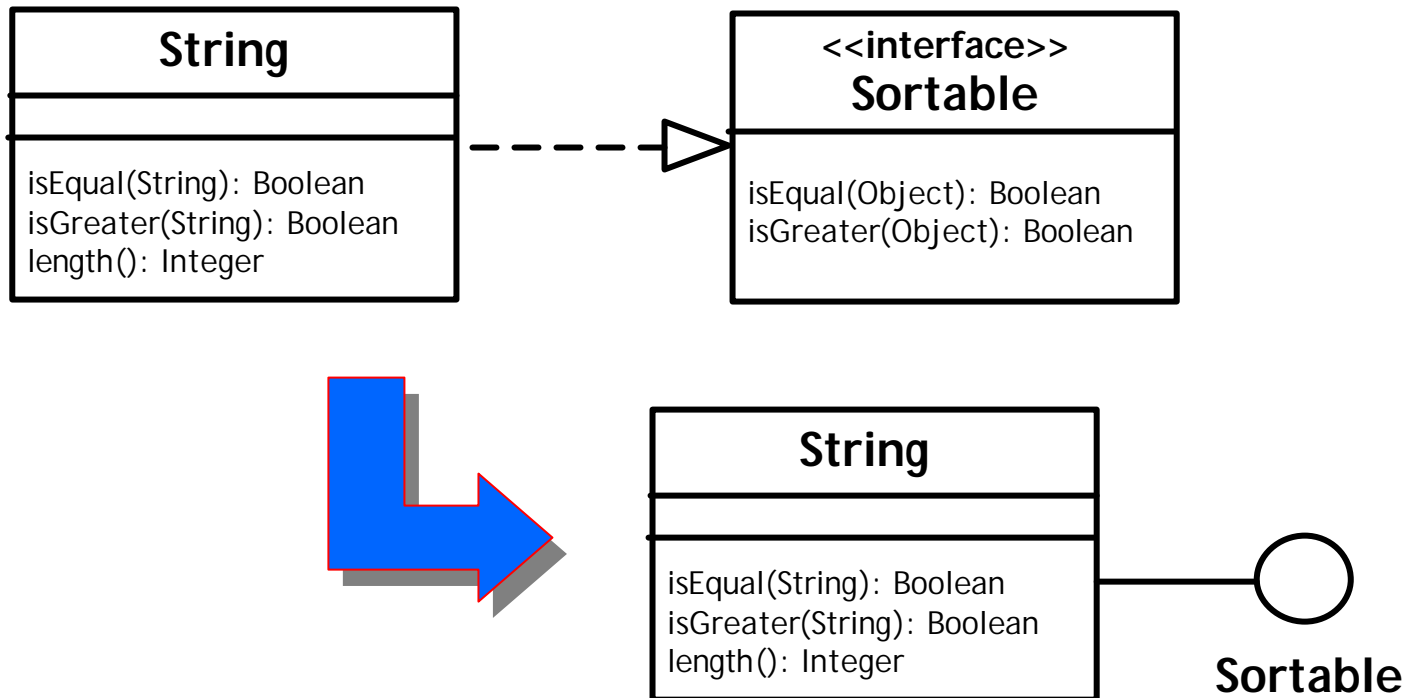
Classi (visibilità)



+ public; **-** private **#** protected

Interfacce

- ✓ Definiscono il comportamento esterno di una classe
- ✓ Descrivono relazione tra classi: una classe viene collegata alle interfacce che essa implementa
- ✓ Interfacce rappresentate con cerchioini
 - notazione complessiva chiamata "lollipop" (lecca-lecca)



Dettagli nella descrizione delle classi

- ✓ UML consente di esprimere graficamente livelli crescenti di dettagli nella descrizione delle classi
- ✓ Questi livelli crescenti di dettagli sono spesso inappropriati o addirittura completamente fuori luogo nella specifica dei requisiti
- ✓ Diventano invece essenziali nella descrizione dell'architettura della soluzione, dove le classi corrispondono esattamente alle classi della soluzione in Java

Dalla specifica all'implementazione

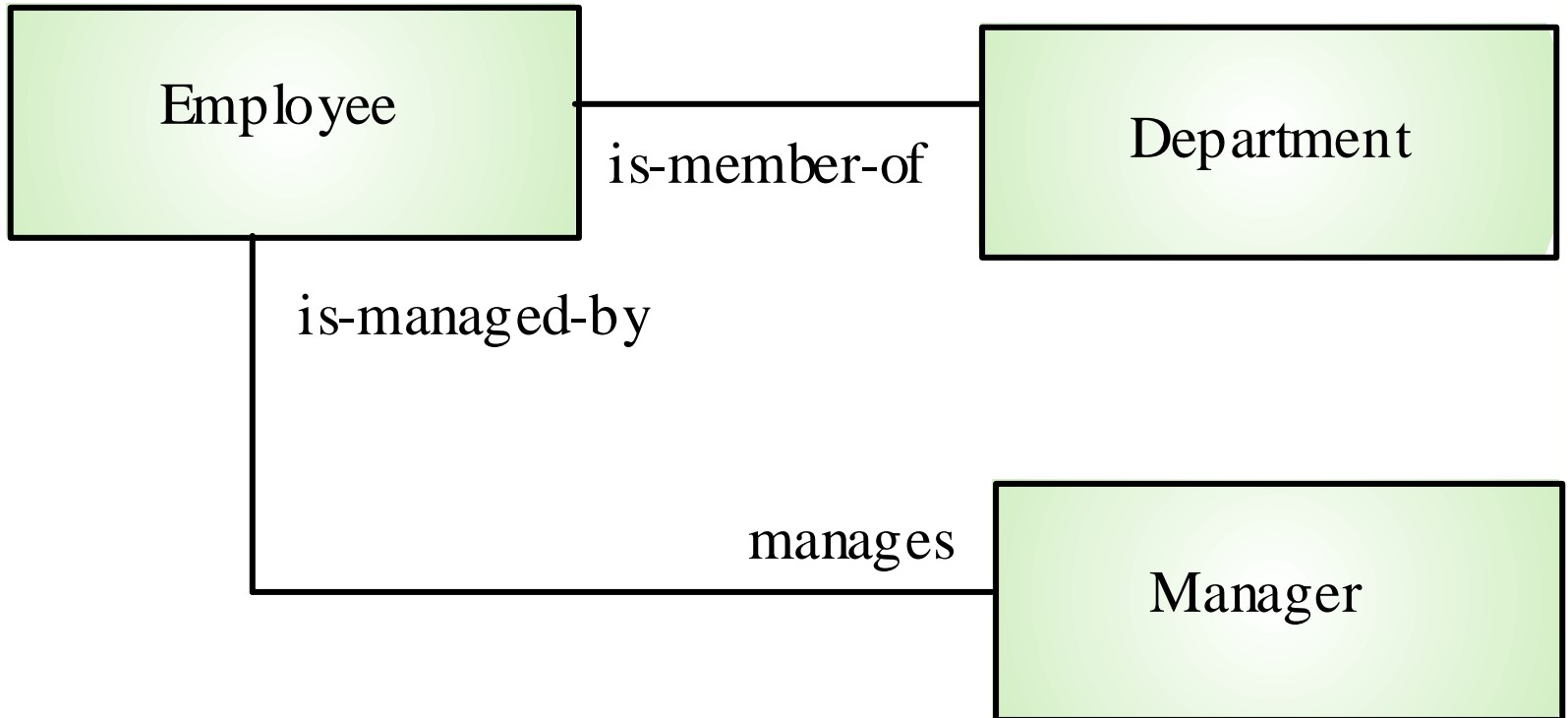
- ✓ UML fornisce una notazione diagrammatica che può essere usata ad ogni livello del processo
- ✓ La scelta dei dettagli a cui scendere dipende dal livello e dallo scopo della descrizione
- ✓ Quando siamo a livello di specifica le classi descrivono le entità logiche in gioco, NON le classi dell'implementazione in Java
- ✓ A livello di specifica ci si ferma spesso alla pura evidenziazione delle classi e delle associazioni
 - dettagli sugli attributi, sui metodi e sulla visibilità potrebbero essere irrilevanti

Associazioni

- ✓ Un'associazione indica una relazione tra classi
 - ad esempio persona che lavora per azienda
- ✓ Un'associazione deve avere un nome, solitamente un verbo (un'etichetta al centro della linea che definisce l'associazione)
 - Associazioni possono essere annotate con informazioni che descrivono l'associazione
 - Nessun significato formale...
 - Associazioni possono anche indicare un oggetto è associato con uno (o più) metodi o attributi di altri oggetti

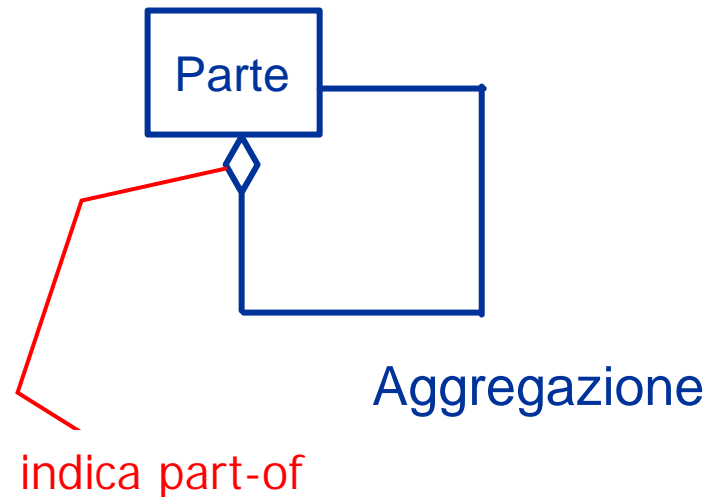
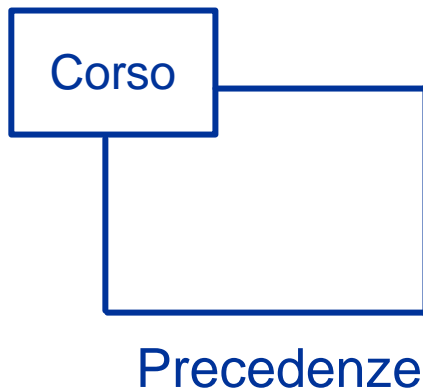


Esempio



Associazioni riflesse

- ✓ Un'associazione è riflessiva se coinvolge oggetti della stessa classe
- ✓ Indicano oggetti multipli della stessa classe che sono in relazione fra loro



Molteplicità

- ✓ La molteplicità dice:
 - Se l'associazione è obbligatoria oppure no
 - Il numero minimo e massimo di oggetti che possono essere relazionati ad un altro oggetto



Molteplicità (cont.)

1	<input type="text"/>	Esattamente 1
*	<input type="text"/>	zero o più
0..1	<input type="text"/>	zero o uno
m..n	<input type="text"/>	entro gli estremi specificati

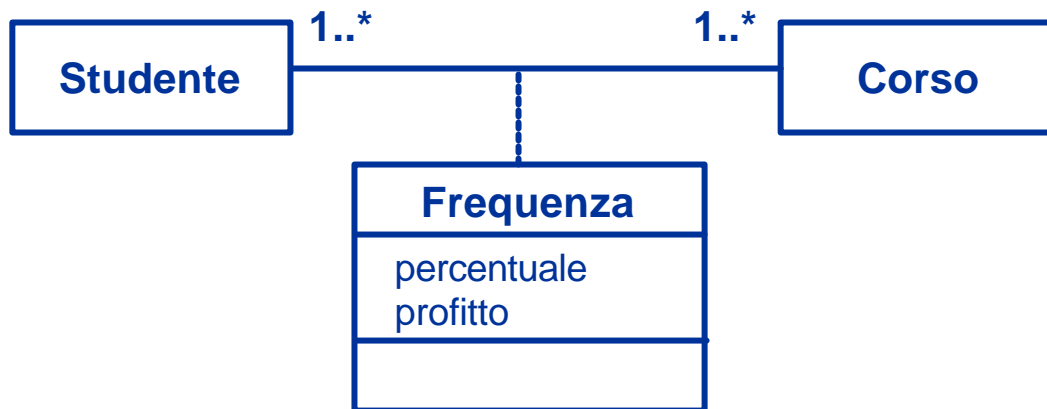
Ruolo

- ✓ Definisce il ruolo svolto nell'associazione



Attributi

- ✓ Alcune proprietà potrebbero appartenere all'associazione e non alle parti coinvolte



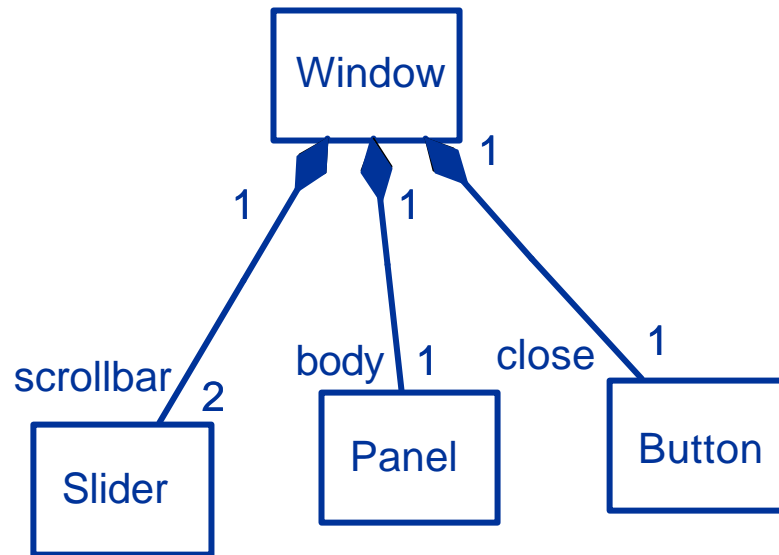
Aggregazioni

- ✓ Le aggregazioni sono una forma particolare di associazione. Una parte è in relazione con un oggetto (**part-of**)

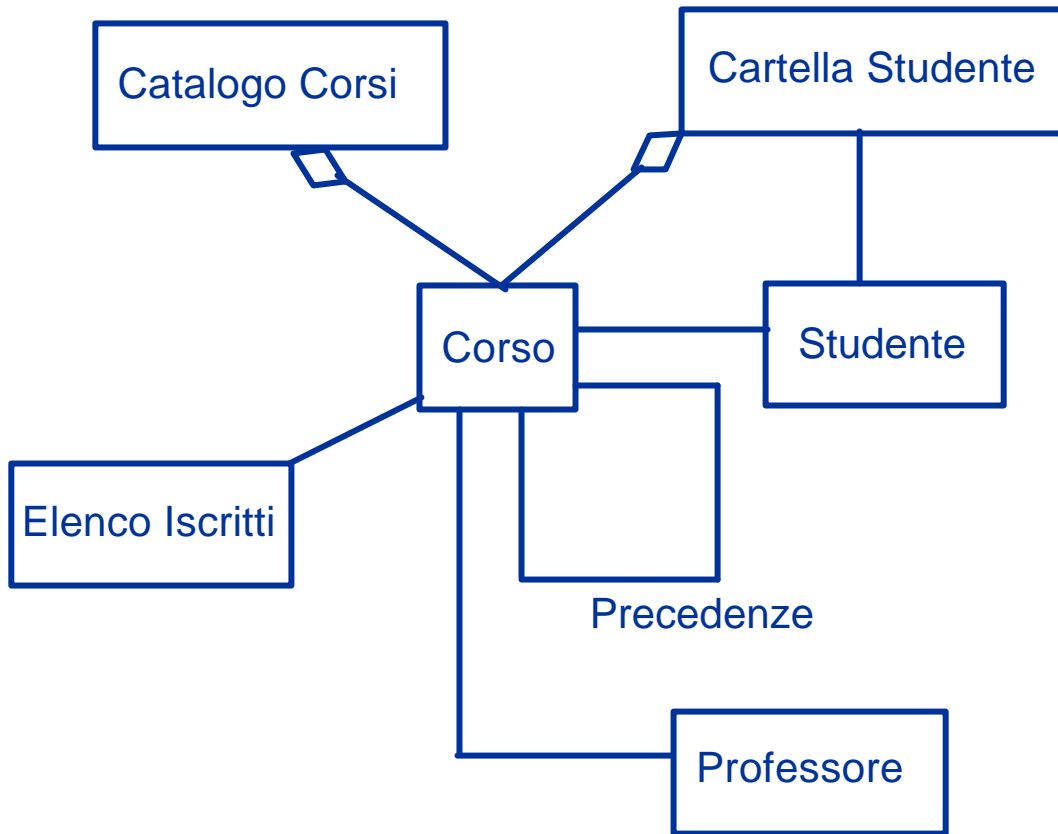


Composizioni

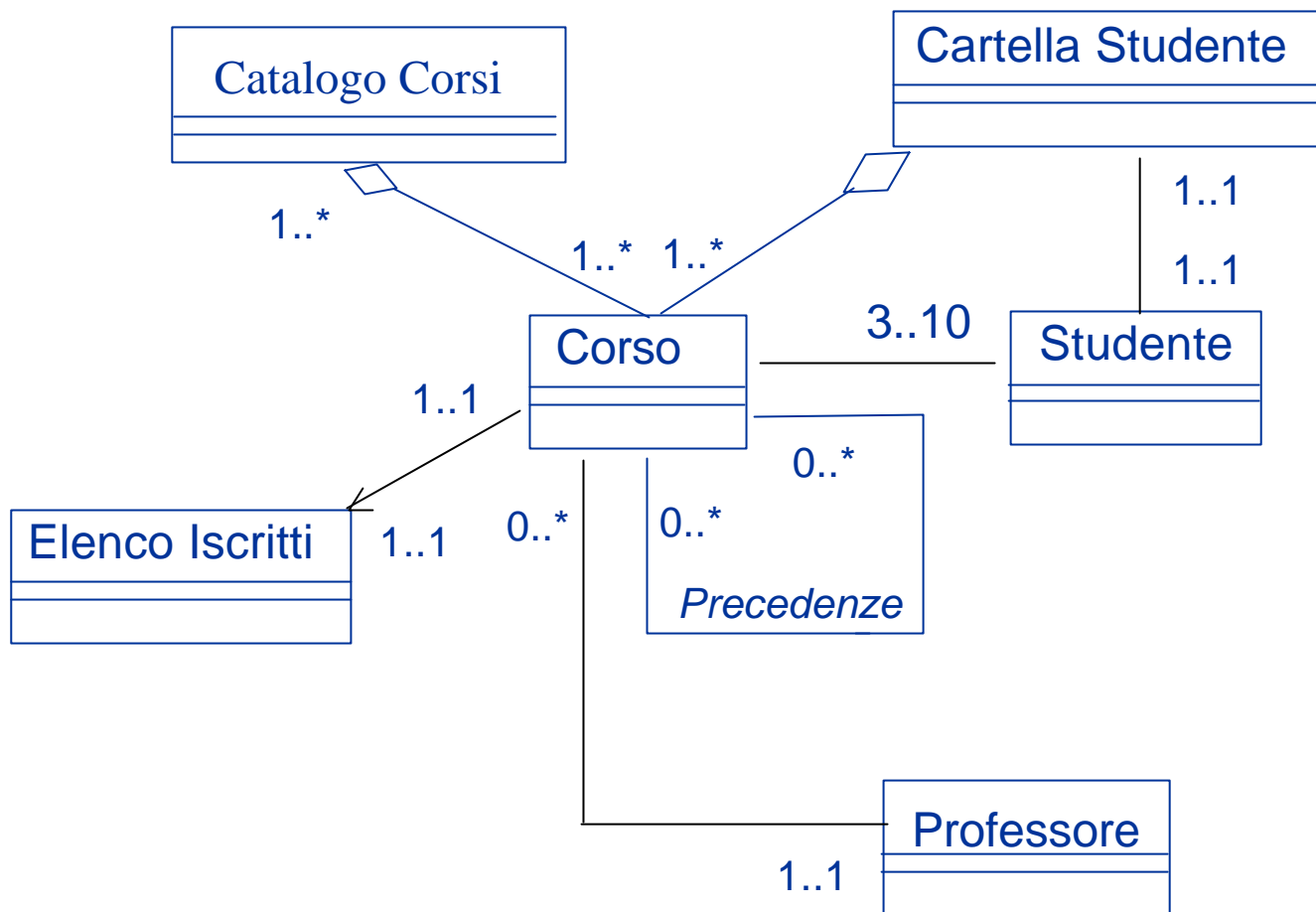
- ✓ Una relazione di composizione è un'aggregazione forte
 - Le parti componenti non esistono senza il contenitore
 - Creazione e distruzione avvengono nel contenitore
 - I componenti non sono parti di altri oggetti
 - notazione: rombi neri



Registrazione Corsi

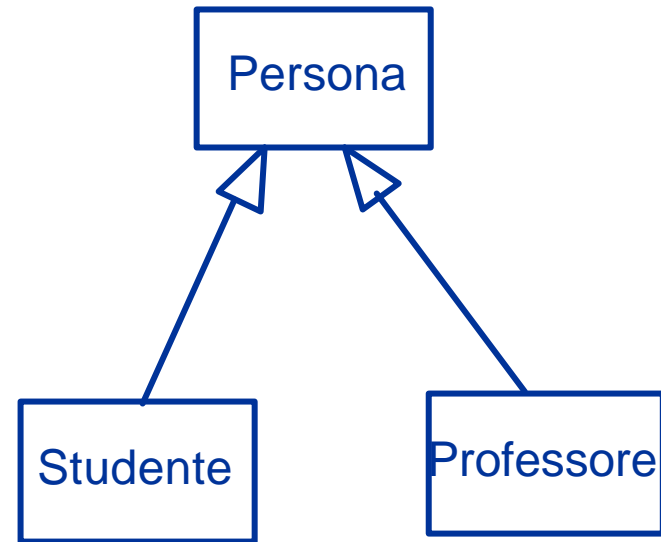


Soluzione con molteplicita'

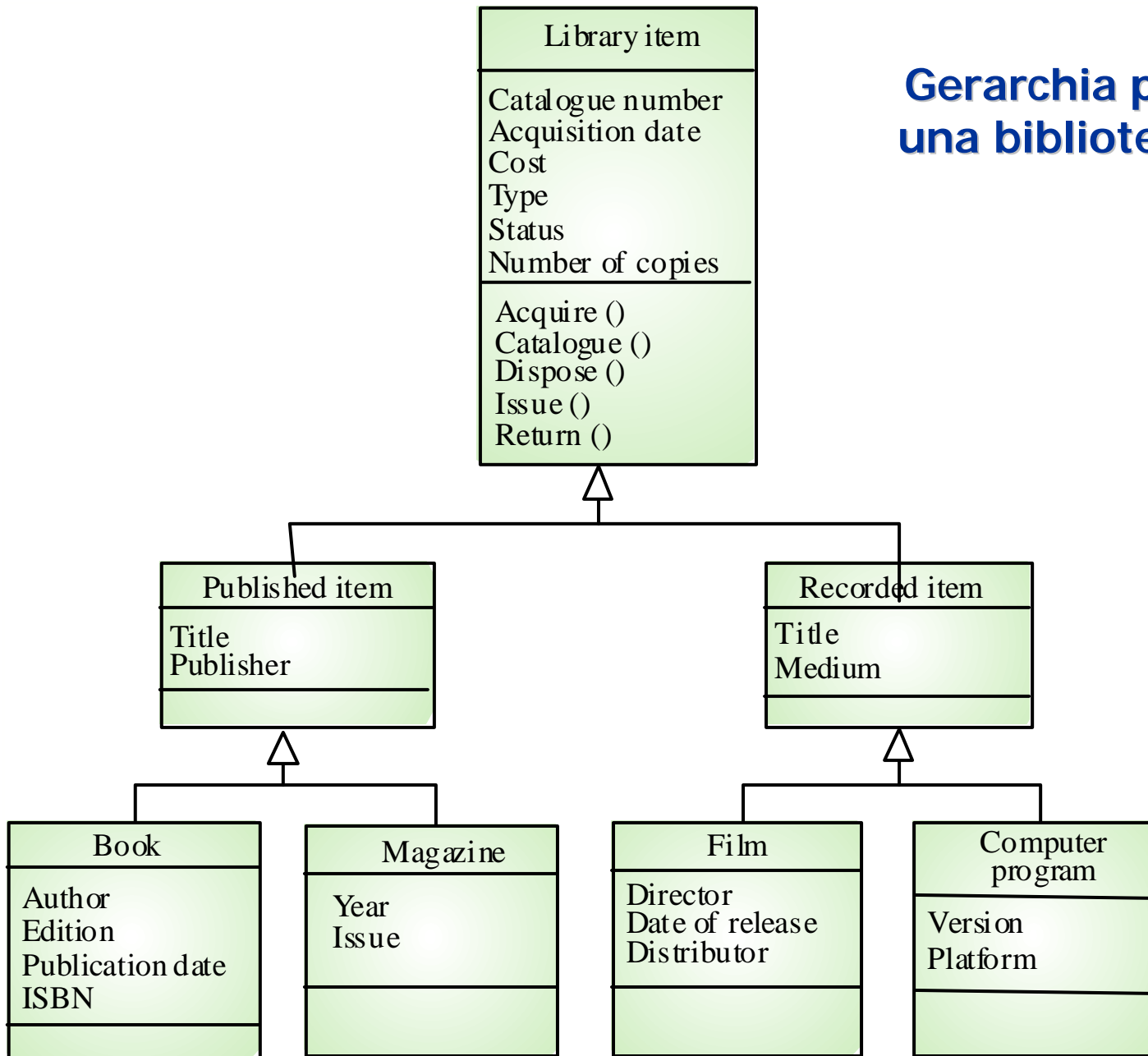


Ereditarietà (Generalizzazione)

- ✓ Esplicita eventuali comportamenti comuni
- ✓ È possibile dover:
 - Aggiungere nuove proprietà alle classi
 - Ridefinire/modificare operazioni esistenti
- ✓ Definisce gerarchia:
 - Le classi in cima alla gerarchia hanno caratteristiche comuni a tutte le classi
 - Le classi più in basso ereditano attributi e servizi dalle classi superiori, ma possono essere specializzate se necessario



Gerarchia per una biblioteca

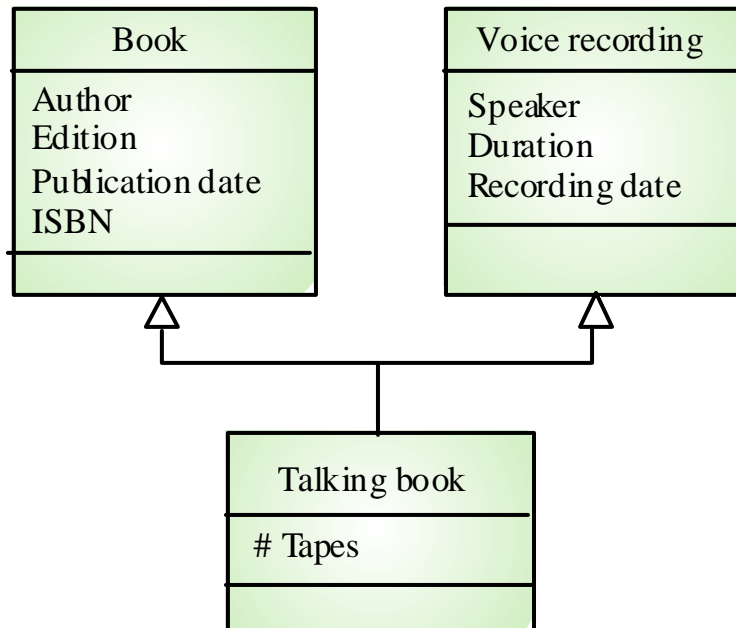


Vantaggi ereditarietà

- ✓ Meccanismo di astrazione utile per classificare entità e trovare relazioni fra entità simili
 - Es.: libro, rivista, film, ecc. : hanno caratteristiche comuni, fattorizzabili in una classe `LibraryItem`, da cui le altre ereditano.
- ✓ Meccanismo di riuso, sia per l'analisi del dominio, che a livello di design e di programmazione
- ✓ Il grafo dell'ereditarietà costituisce una sorgente di informazione organizzativa sul dominio dell'applicazione e sui sistemi realizzati, utile anche documentazione
- ✓ Problema di leggibilità: classi non sono entrocontenute: occorre leggere anche le superclassi per capirle...

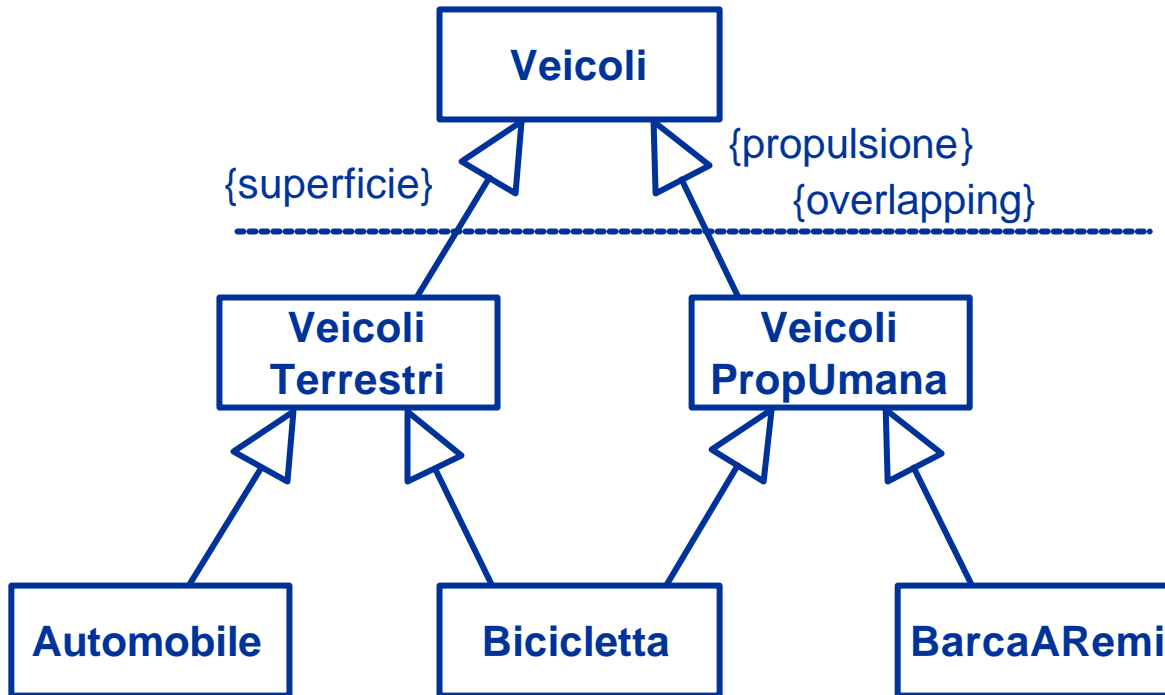
Ereditazione multipla

- ✓ Possibilità di ereditare da più classi
- ✓ Può portare a conflitti fra attributi o servizi con lo stesso nome ereditati da classi diverse



Ereditarietà

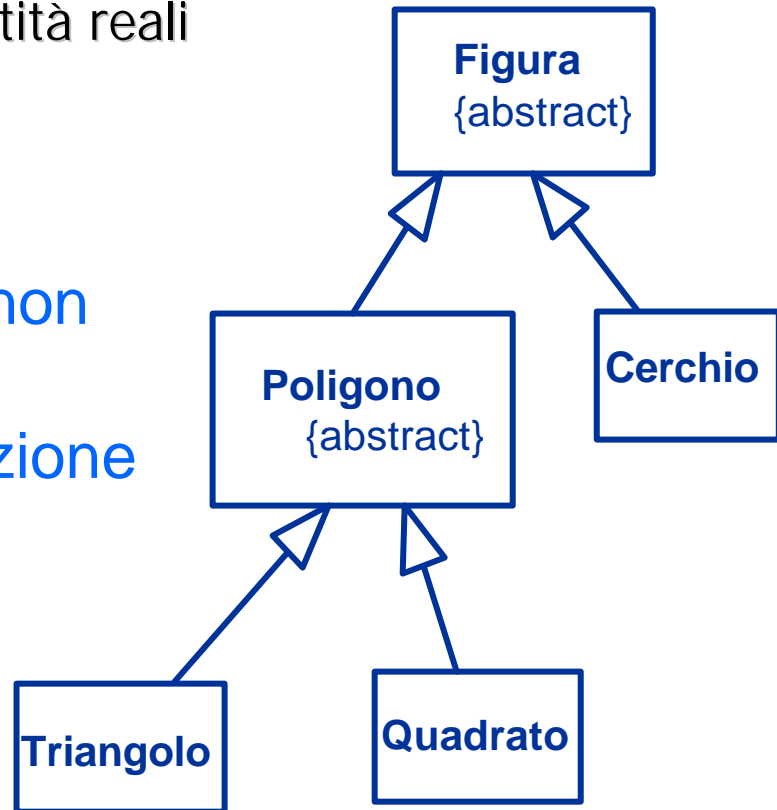
(Sottoclassi non Disgiunte)



Classi Astratte

- ✓ Una classe e' astratta quando non puo'essere istanziata
 - Classi astratte usate solo per definire generalizzazioni, che non corrispondono a entità reali
 - notazione: {abstract}

N.B. si tratta di costrutti utili nel descrivere la soluzione, non il problema!
Architettura dell'implementazione vs specifica dei requisiti



Esempio

- ✓ A simple system:
- ✓ a *controller* uses a *sensor* with an analog-to-digital *converter* to acquire information.
- ✓ The controller then uses an *actuator* to affect its environment.
- ✓ The system may use a temperature sensor to activate either a heater or a fan for closed-loop control.
- ✓ Additionally, it may use a pressure sensor to provide information about a gas line that it uses to control a valve with a stepper motor.

alcune classi per l'esempio

Temperature Sensor
temperature calibration constant
acquire() set calibration()

Stepper Motor
position
step forward() step backward() park() power()

RS232 Interface
data to be transmitted data received baud rate parity stop bits start bits last error
send message() receive message() set comm parameters() pause() start() get error() clear error()

Class Diagram per l'esempio

