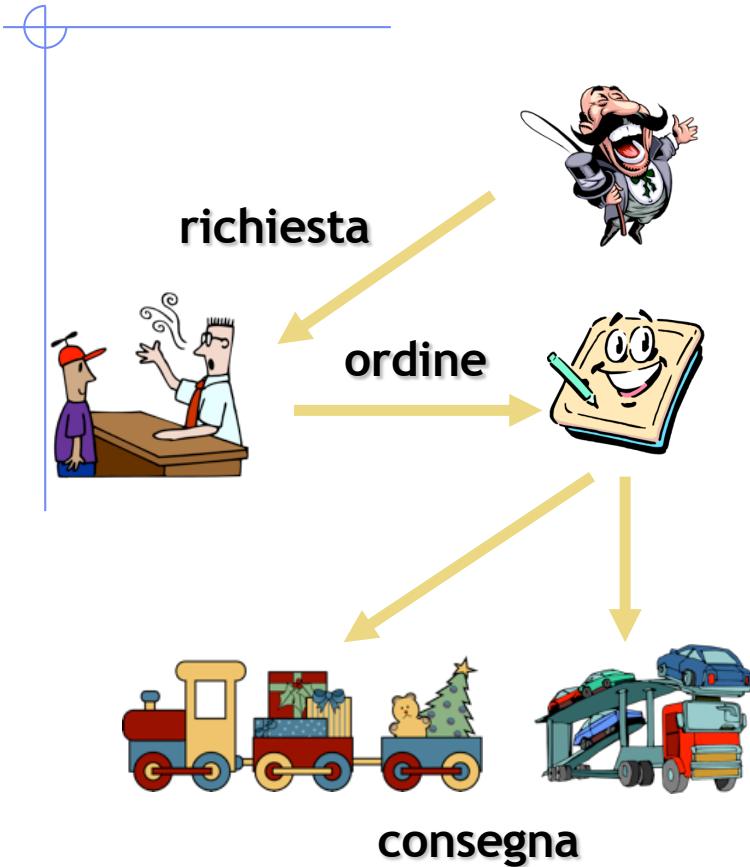


Unified Modeling Language (UML): una breve introduzione

Gian Pietro Picco

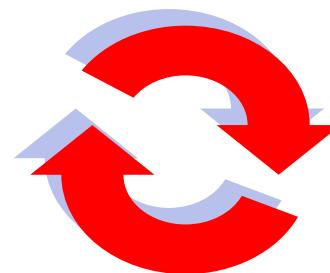
Modellazione visuale



Business Process

“Un modello cattura le parti essenziali di un sistema”

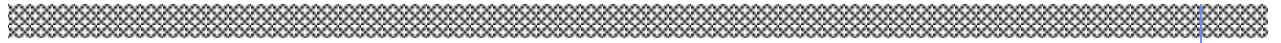
James Rumbaugh



Computer System

Perché UML

- ✖ È il linguaggio visuale standard (OMG) per definire, progettare, realizzare e documentare i sistemi software ad oggetti
- ✖ Riunisce molte proposte esistenti (Booch, Rumbaugh e Jacobson)
- ✖ Copre l'intero processo di produzione
- ✖ È sponsorizzato dalle maggiori industrie produttrici di software



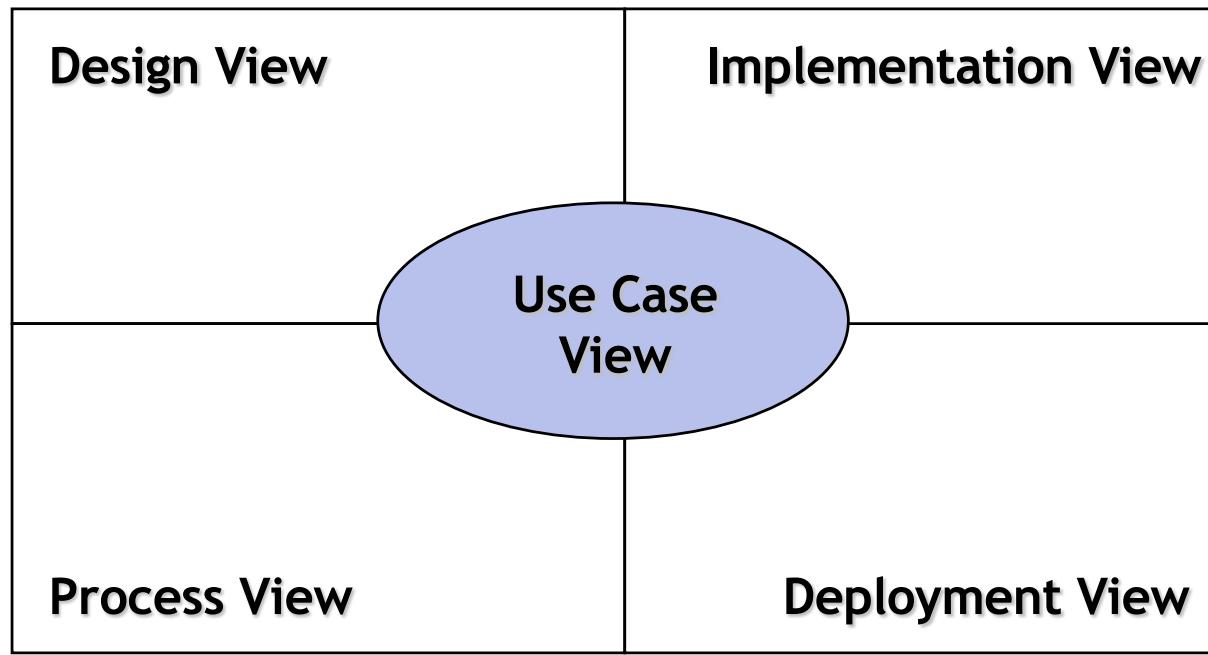
Perché UML

- ✖ Riunisce aspetti dell'ingegneria del software, delle basi di dati e della progettazione di sistemi
- ✖ È indipendente da qualsiasi linguaggio di programmazione
- ✖ È utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni
- ✖ È estendibile per modellare meglio le diverse realtà applicative

Modelli

- ✖ I modelli rappresentano il linguaggio dei progettisti
 - I modelli rappresentano il sistema da costruire o costruito
 - I modelli sono un veicolo di comunicazione
 - I modelli descrivono in modo “visuale” il sistema da costruire
- ✖ I modelli sono uno strumento per gestire la complessità
- ✖ I modelli consentono di analizzare caratteristiche particolari del sistema

4+1 Viste



P. Kruchten, “The 4+1 View Model of Software Architecture”.
IEEE Software 12 (6), pages 42-50, november 1995

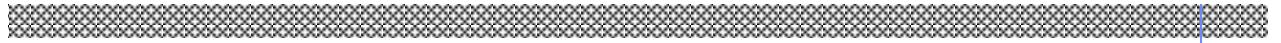
Diagrammi UML

*Viste statiche

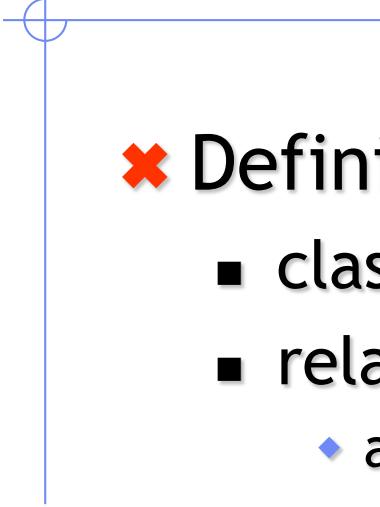
- Use Case Diagrams
- Class Diagrams
- Object Diagrams
- Component Diagrams
- Deployment Diagrams

*Viste dinamiche

- Sequence Diagrams
- Collaboration Diagrams
- Statechart Diagrams
- Activity Diagrams



Class Diagram

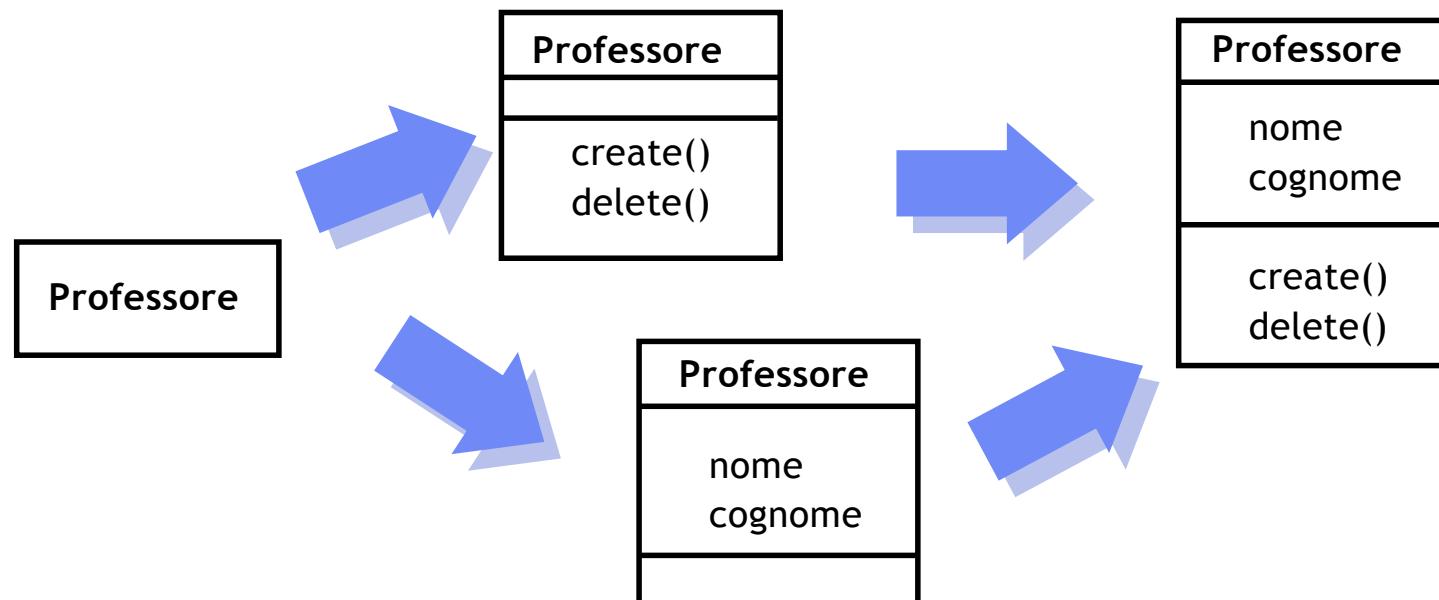


- ✖ Definiscono la visione statica del sistema
 - classi
 - relazioni tra classi
 - ◆ associazione (uso)
 - ◆ generalizzazione (ereditarietà)
 - ◆ aggregazione (contenimento)
- ✖ È forse il modello più importante perché definisce gli elementi base del sistema

Classe

- * In UML è composta da tre parti

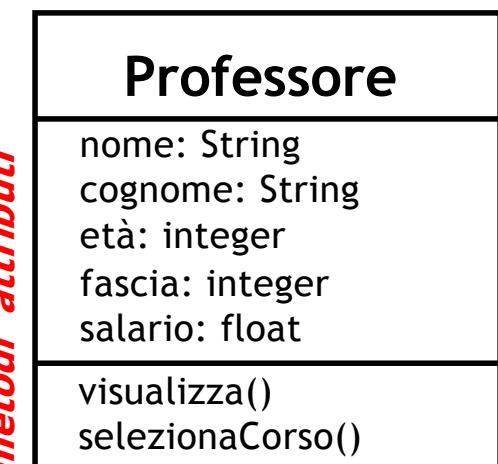
- nome
- attributi (lo stato)
- metodi (il comportamento)



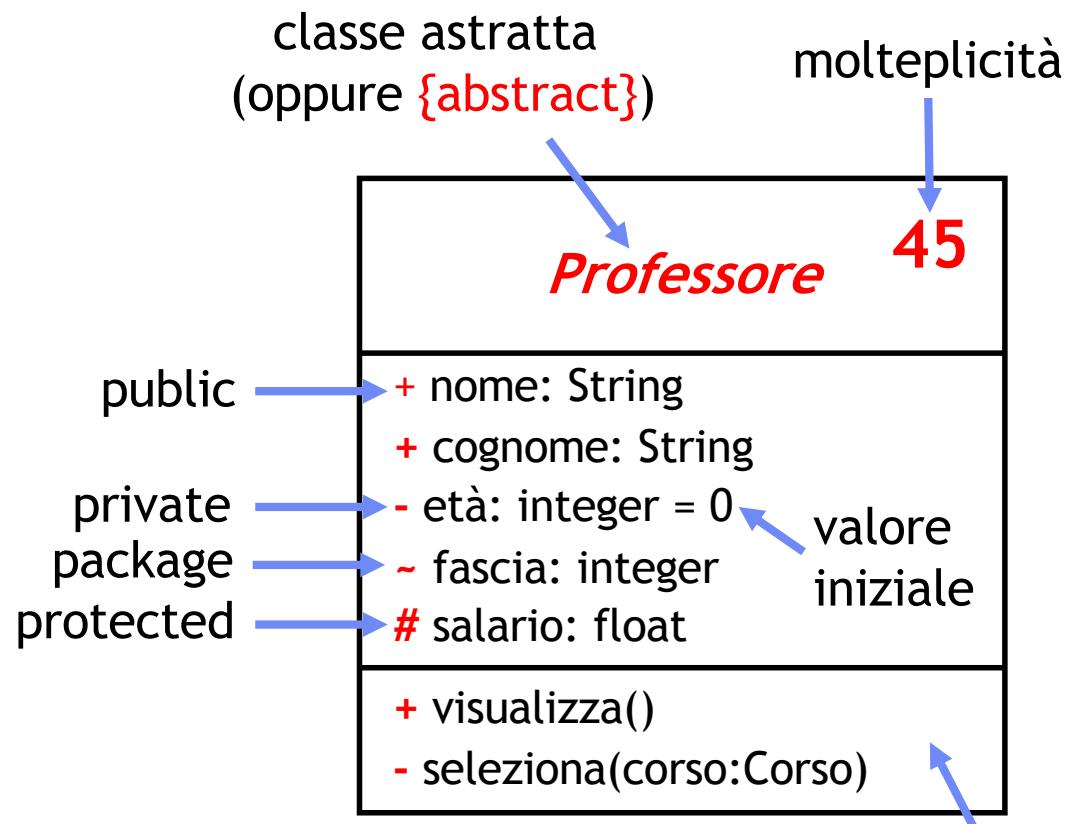
Attributi

- ✖ Un attributo è una caratteristica della classe
 - Gli attributi non hanno una loro «identità»
- ✖ Spesso sono derivati da «nomi» (es. in una specifica testuale) che non sono diventati classi durante la progettazione
- ✖ Determinati da conoscenza del dominio applicativo
 - Persona (ambito bancario)
 - ◆ nome, cognome, codiceFiscale, numeroConto
 - Persona (ambito medico)
 - ◆ nome, cognome, allergie, peso, altezza

Classi: notazione



metodi attributi



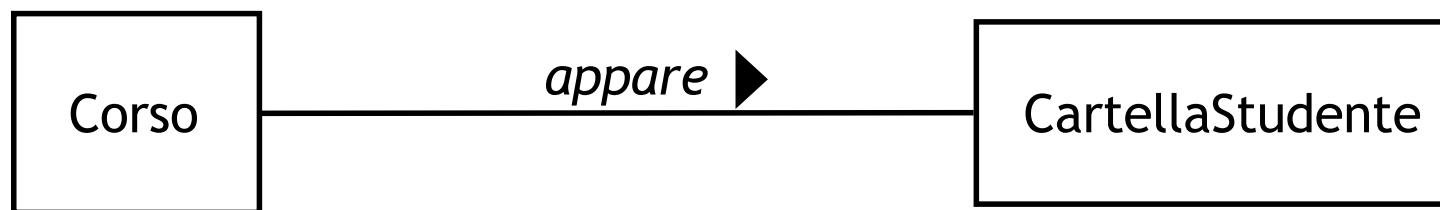
Method ::= [visibility] name [(parameter list)] [: return type]

Parameter ::= [direction] name: type [= default-value]

se sottolineati
metodi e attributi si
intendono static

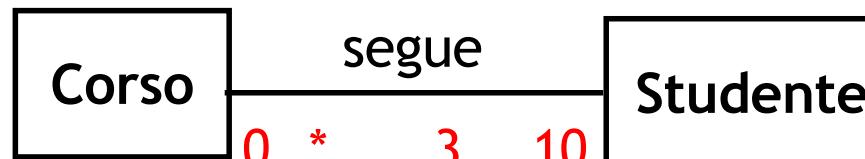
Associazione

- ✖ Indica una relazione tra classi
 - un «canale di comunicazione» bidirezionale
- ✖ Deve avere un nome, solitamente un verbo
 - etichetta al centro della linea che definisce l'associazione
- ✖ È possibile specificare una direzione, se non evidente



Molteplicità

- ✖ Definisce il numero di istanze che prendono parte alla relazione, specificando
 - se l'associazione è obbligatoria oppure no
 - il numero minimo e massimo di oggetti che possono essere relazionati ad un altro oggetto



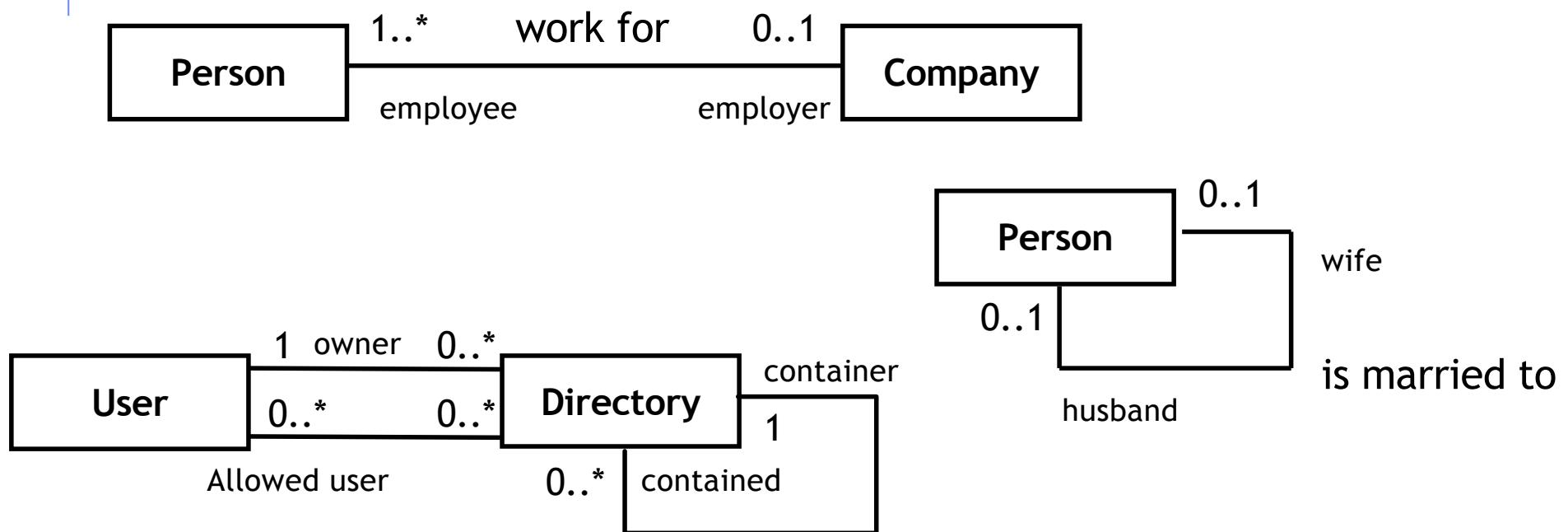
ogni studente
può seguire nessun corso,
oppure un numero arbitrario

ogni corso può avere
un minimo di 3 e
un massimo di 10 studenti

- ✖ esattamente uno: 1
- ✖ zero o uno: 0..1
- ✖ molti: 0..*
- ✖ uno o più: 1..*
- ✖ un numero specifico: 7
- ✖ un intervallo: 4..15
- ✖ una lista: 0..1, 3..4, 6..*

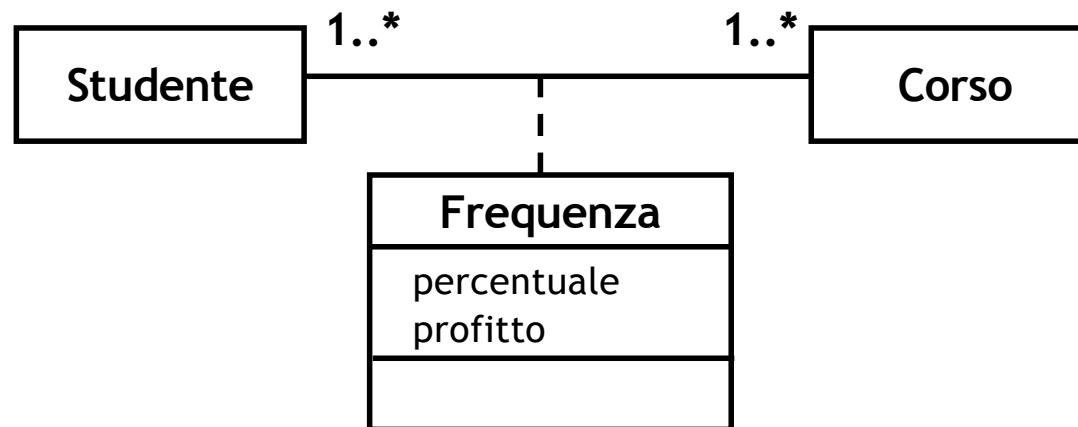
Ruolo

- ✖ Definisce il ruolo svolto nell'associazione
- ✖ Utile per
 - Auto-associazioni
 - Associazioni multiple tra due classi



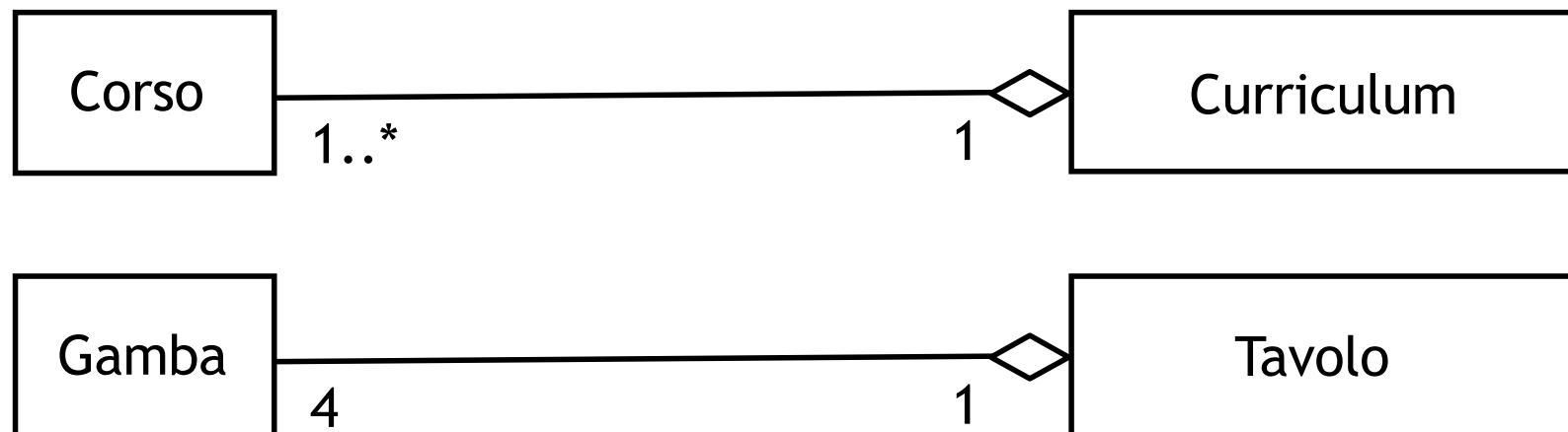
Classi associazione

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



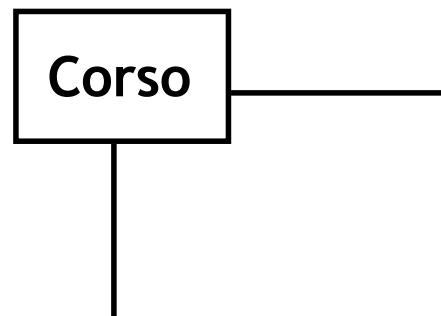
Aggregazione

- ✖ È una forma particolare di associazione ...
- ✖ ... in cui una parte è in relazione con un oggetto (**part-of**)

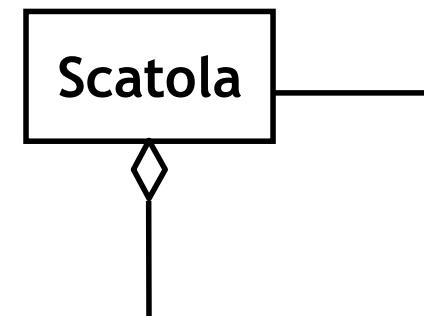


Associazione riflessiva

- * Un'associazione è tale se coinvolge oggetti della stessa classe
 - oggetti multipli della stessa classe sono in relazione fra loro



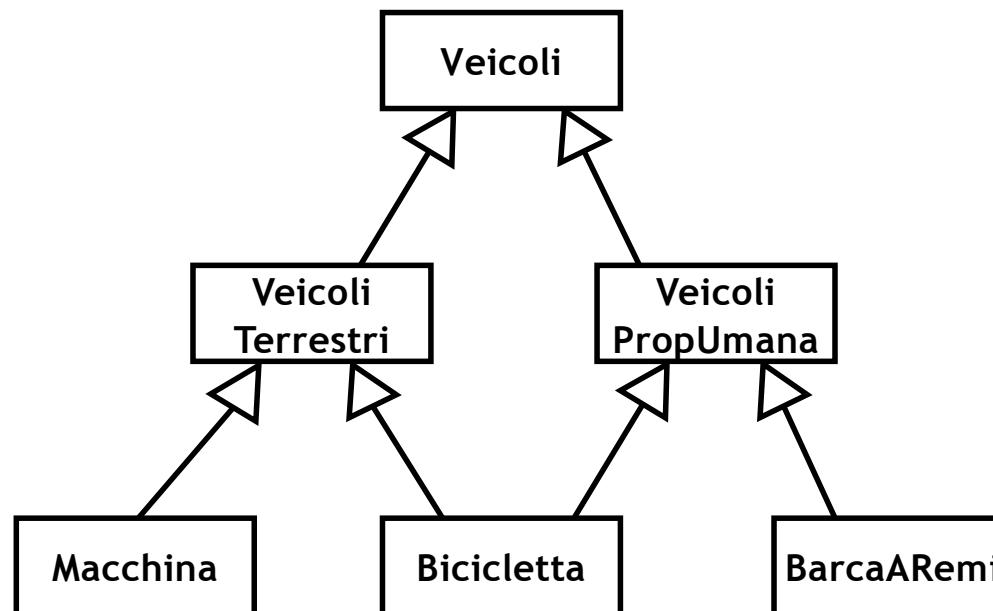
ha precedenza



contiene

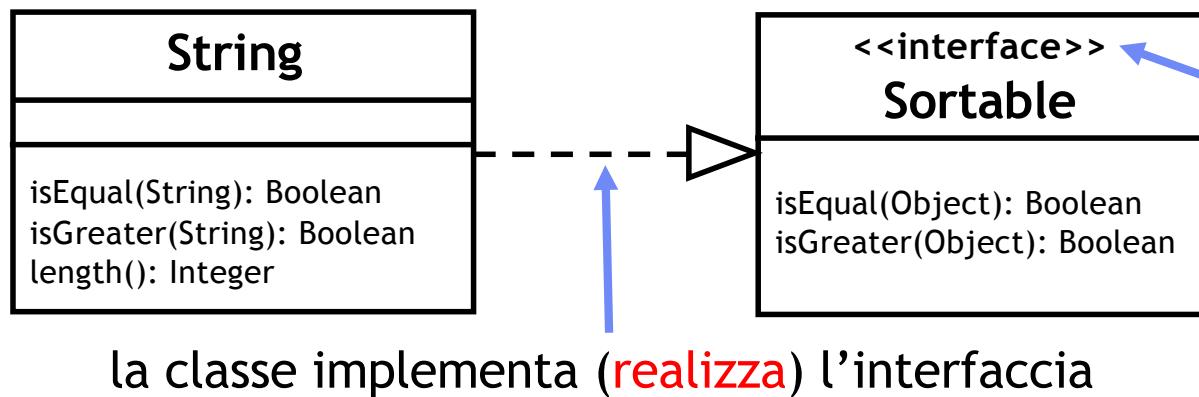
Ereditarietà (Generalizzazione)

- ✖ Esplicita eventuali comportamenti comuni
- ✖ È possibile:
 - aggiungere nuovi attributi alle classi
 - ridefinire/modificare metodi esistenti

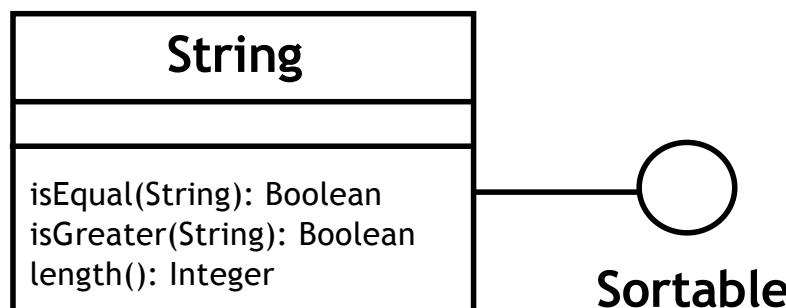


Interfaccia: definizione

- ✖ Specifica il comportamento di una classe senza darne l'implementazione

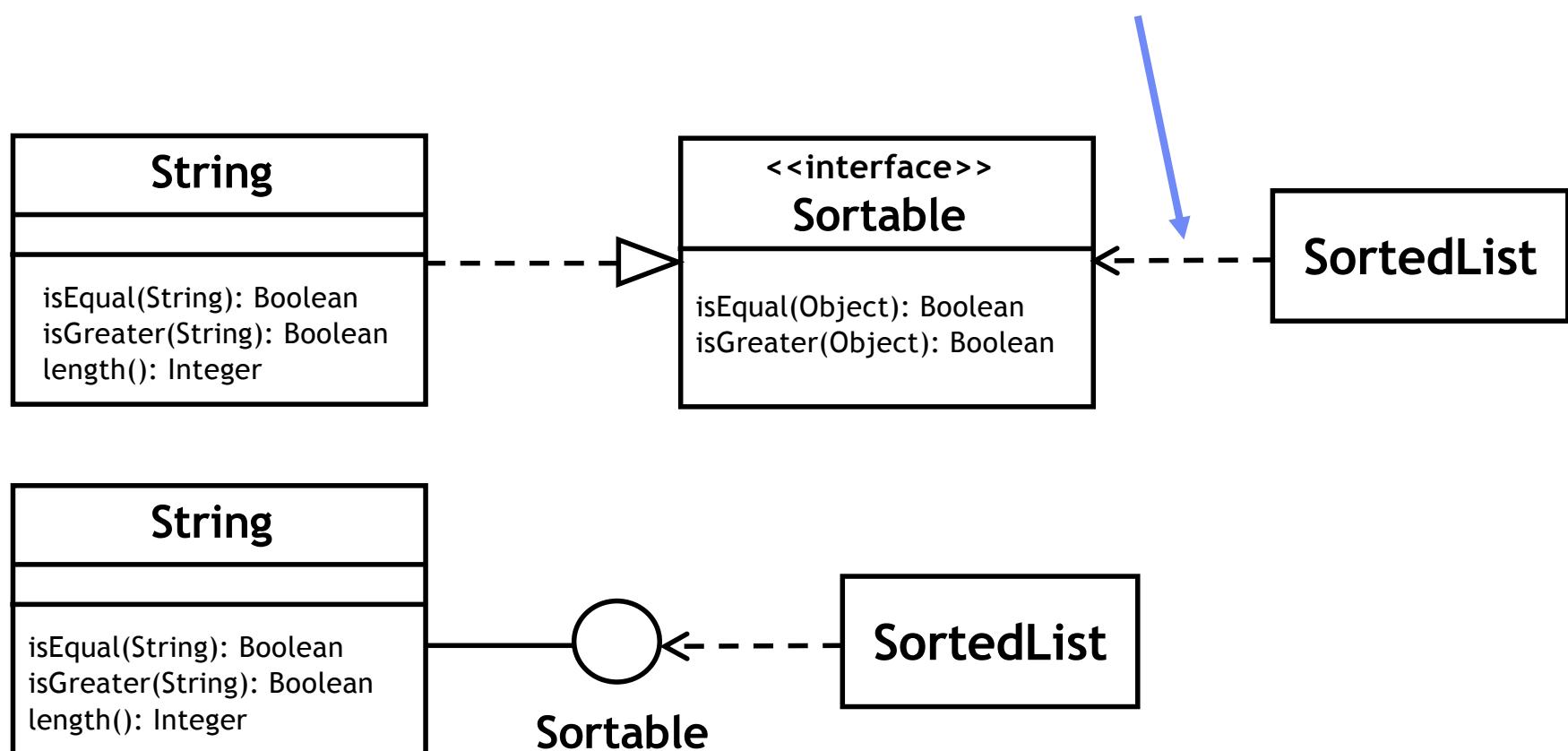


stereotipo:
altro esempio è
`<<utility>>`, indica
che tutti gli attributi
e metodi sono static



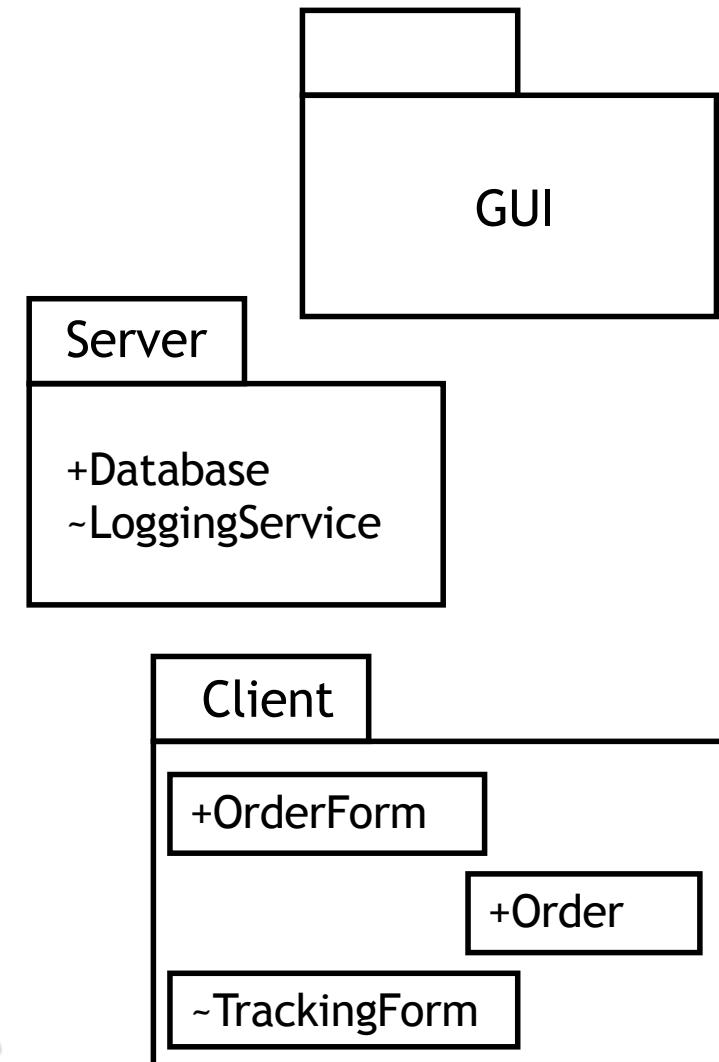
Interfacce e dipendenze

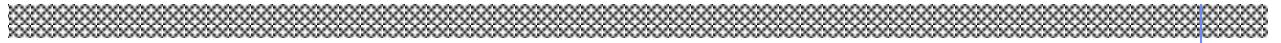
la classe dipende/richiede l'interfaccia
(attenzione a non confondere i due tipi di frecce...)



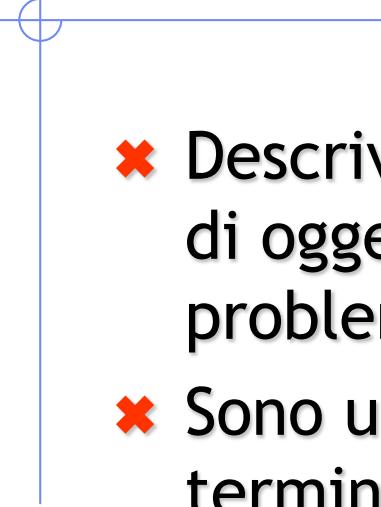
Package

- ✖ I package sono “contenitori” di classi
- ✖ Sistemi complessi (reali) devono essere modellati gerarchicamente
 - Sottosistemi (package) in relazione fra loro
 - Classi all'interno dei package
- ✖ Package
 - Alta coesione all'interno
 - Interfacce precise e limitate





Interaction diagram

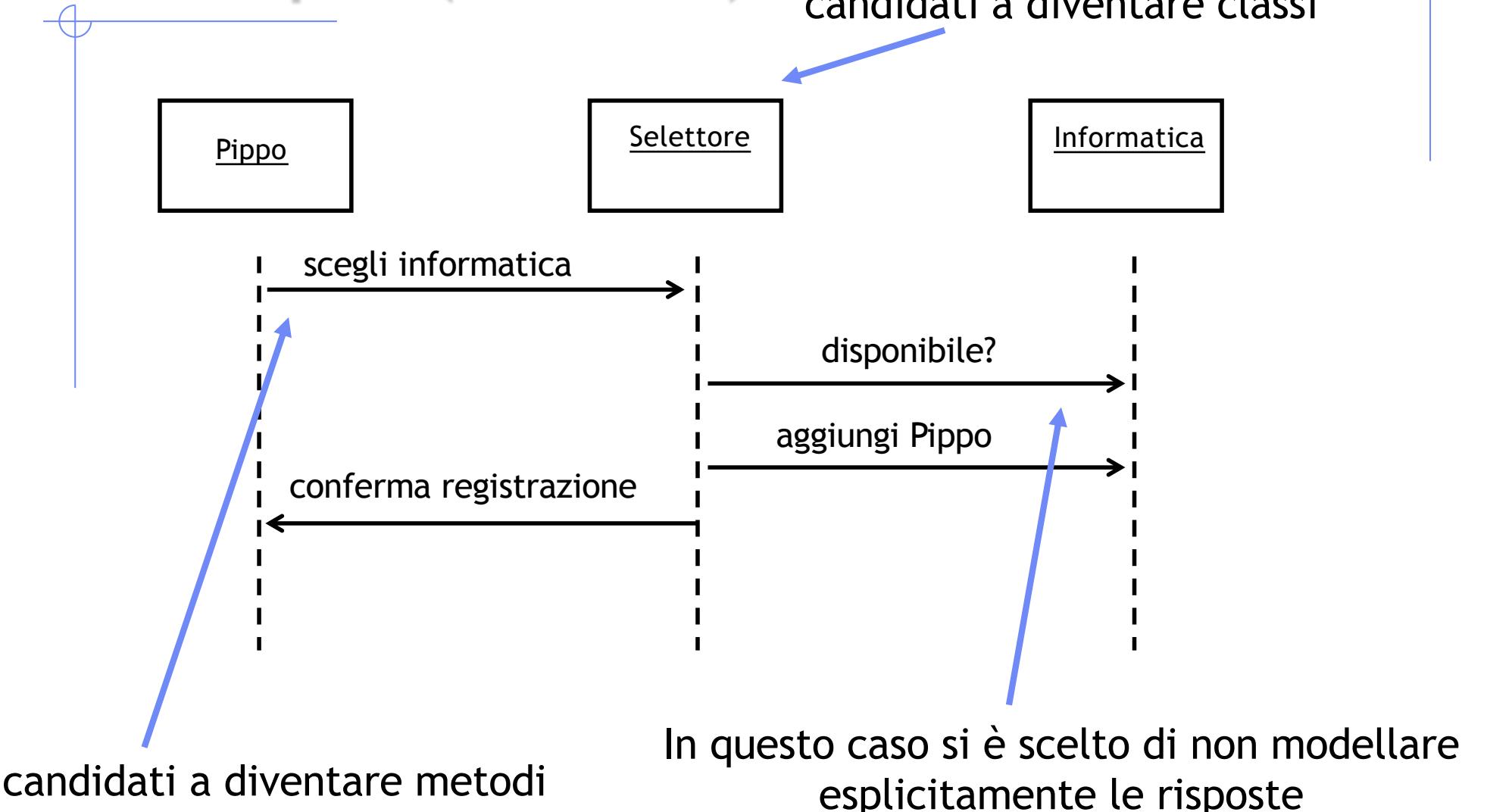


- ✖ Descrivono il comportamento dinamico di un gruppo di oggetti che “interagiscono” per risolvere un problema
- ✖ Sono utilizzati per “formalizzare” gli scenari in termini
 - Entità (oggetti)
 - Messaggi scambiati (metodi)
- ✖ UML propone due diversi tipi di interaction diagram
 - Sequence diagram
 - Collaboration diagram

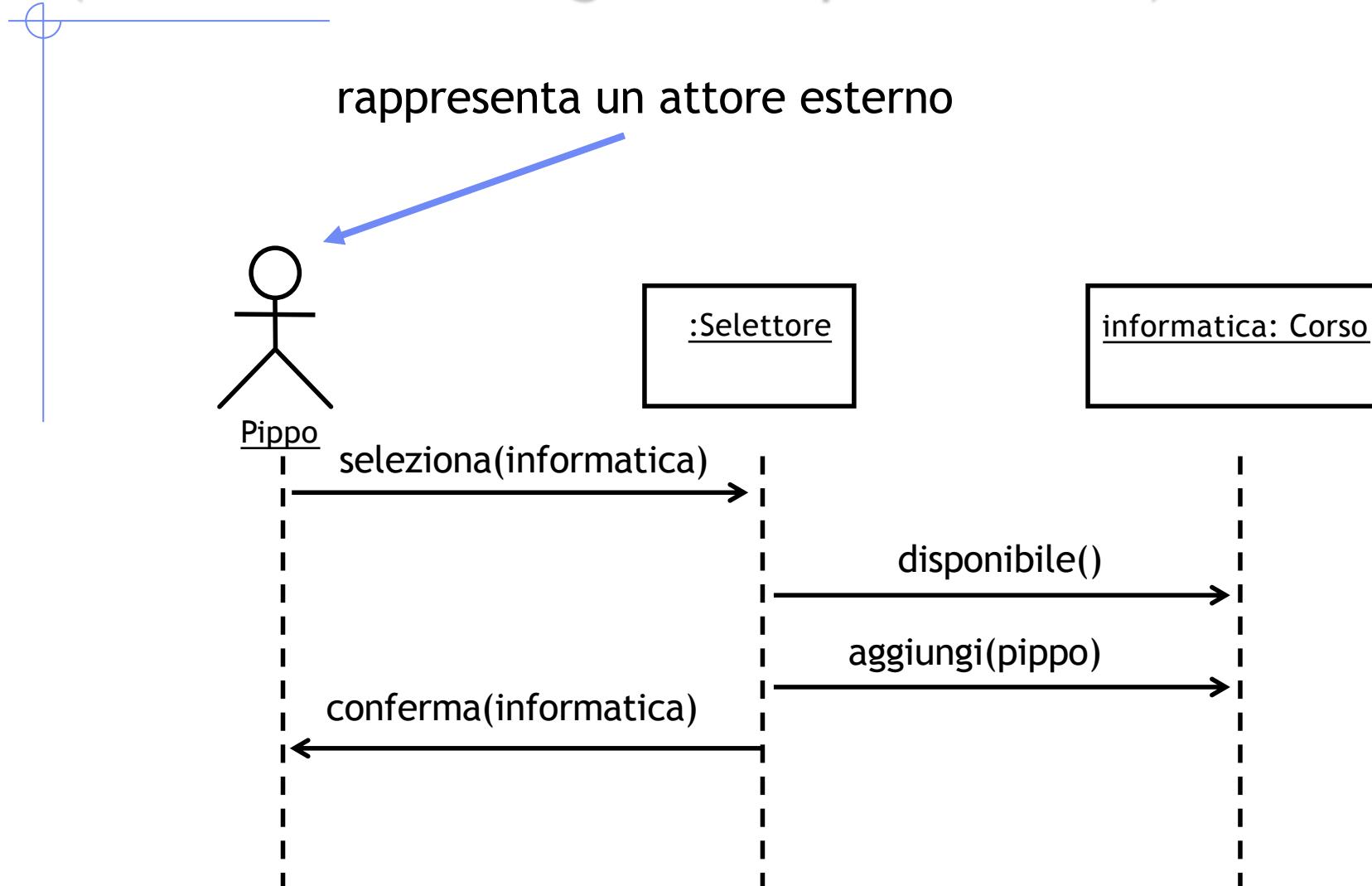
Sequence diagram

- ✖ Evidenziano la sequenza temporale delle azioni
- ✖ Non si vedono le associazioni tra oggetti
- ✖ Usabili in due forme diverse
 - generica: tutte le sequenze (esecuzioni) possibili
 - istanza: una sequenza particolare, consistente con quella generica

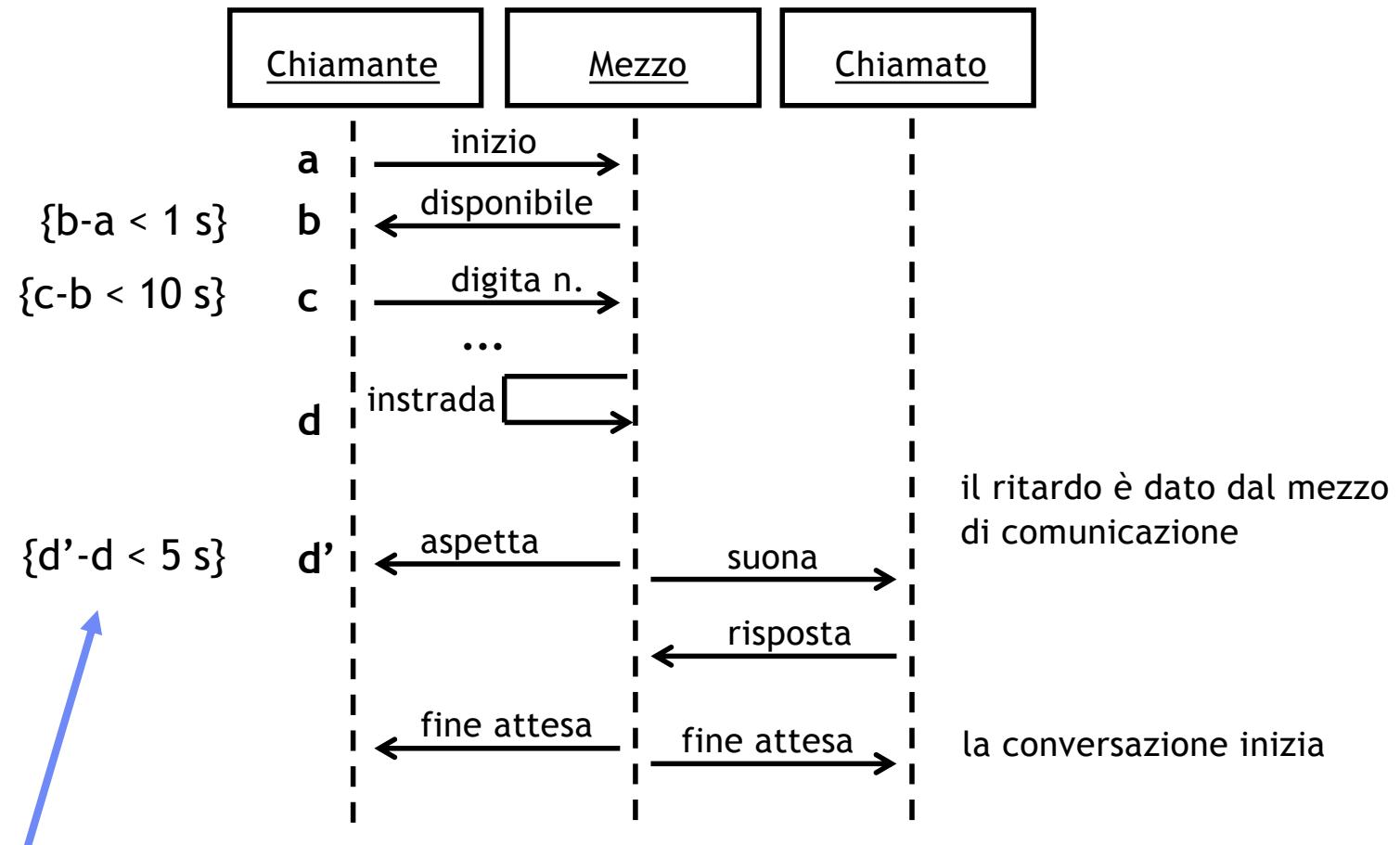
Esempio (istanza)



Esempio (raffinamento diagramma precedente)

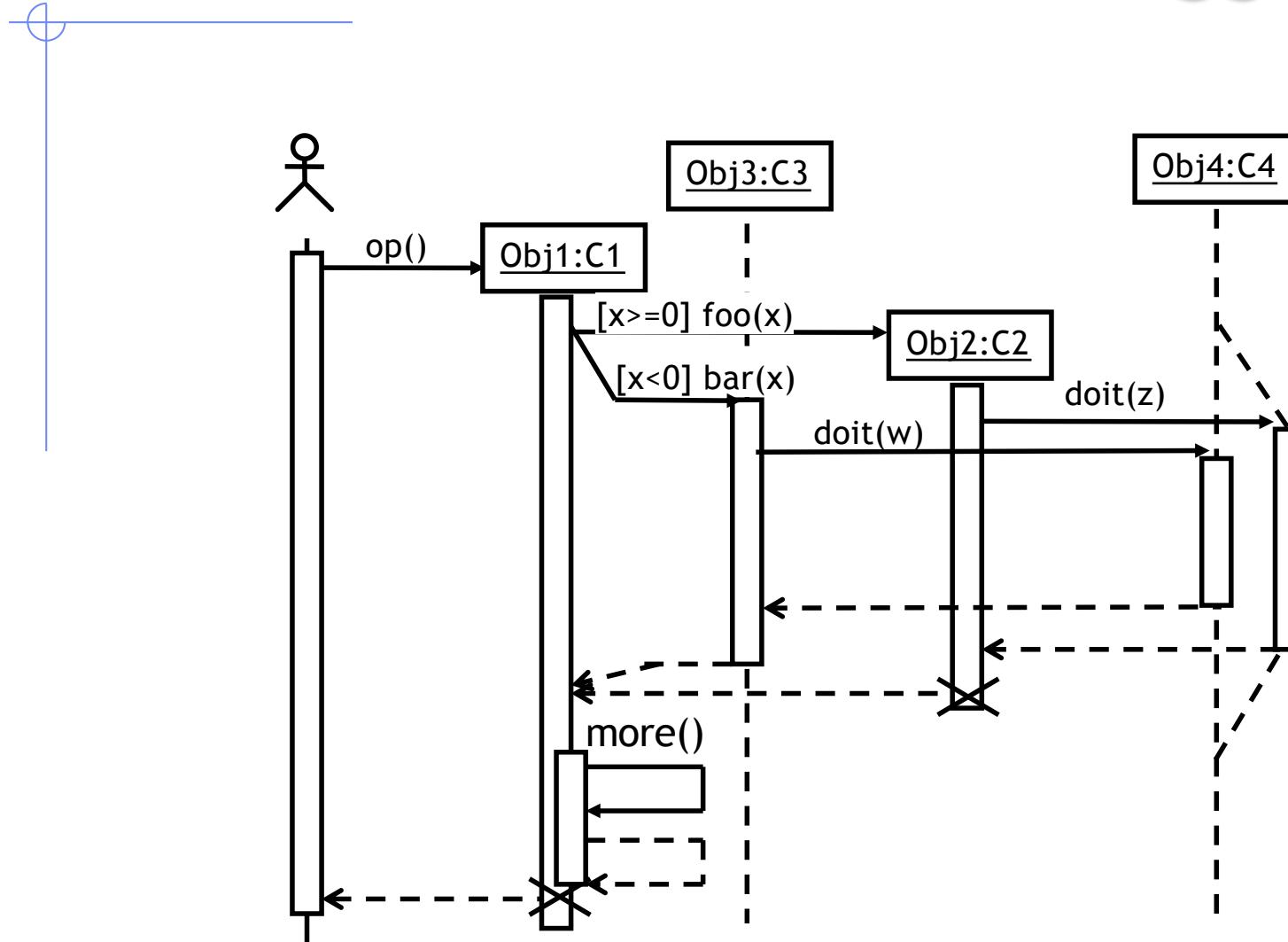


Esempio (chiamata telefonica)



è possibile specificare vincoli
temporali o di altra natura

Sequence diagram: alternative e creazione oggetti



Messaggi

✗ Sincroni

- Il sorgente deve aspettare il destinatario

- ← ▪ Chiamate di procedure o chiamate innestate
- ← ▪ Flusso di controllo piatto

✗ Asincroni

- Il sorgente continua senza aspettare il destinatario
- Utile per sistemi concorrenti (con oggetti attivi)

✗ Condizioni

$[x > 0]$ foo()

✗ Iterazioni

* foo()

✗ Risposte (valore)

- ✗ send
- ✗ create
- ✗ destroy

Esempio

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

Soluzione

