

# Unified Modeling Language (UML)

1

1

## Introduzione



- UML nasce come *standard aperto* dalla collaborazione fra tre dei massimi esperti di OOA: **Grady Booch**, **Ivar Jacobson** e **Jim Rumbaugh** (*i tres amigos*), ed è inteso come sintesi dei molti metodi attualmente usati
- È stato accettato da molti altri esperti del settore, tra cui Coad, Yourdon e Odell, e da tutte le grandi compagnie dell'informatica (tra cui Compaq-Digital, Ericsson, Hewlett-Packard, IBM, Microsoft, Rational Software, ...)
- E' uno **standard dell'OMG** dal 1997
- Esistono potenti strumenti **CASE** per UML. Da un modello UML è possibile generare automaticamente lo "scheletro" del codice di un sistema (le strutture dati complete e i prototipi delle funzioni)
- Microsoft** ha adottato UML come linguaggio standard per la sua libreria di componenti

2

# UML ...

- è un **linguaggio**, non un **metodo** (come quelli di Yourdon e DeMarco, o di Rumbaugh o Jacobson)
- definisce una notazione standard, basata su un **metamodello** integrato degli “oggetti” che compongono un sistema software
- non prescrive una sequenza di processo, cioè non dice “prima bisogna fare questa attività, poi quest’altra”
- quindi può essere (ed è) utilizzato da persone e gruppi che seguono metodi diversi (è “indipendente dai metodi ”)
- è un linguaggio non proprietario, standard; i suoi autori non hanno il copyright su UML
- la versione diventata standard OMG ha ricevuto i contributi di molti altri metodologi e delle più importanti società di software mondiali
- la sua **evoluzione** è a carico dell’OMG, e soggetta a procedure ben definite per ogni cambiamento. Versione attuale: 2.5 (2013)

3

## Generalità

- UML fornisce i costrutti per le seguenti fasi dello sviluppo dei sistemi software:
  - ⇒ Analisi dei requisiti tramite i casi d’uso
  - ⇒ Analisi e progetto OO
  - ⇒ Modellazione dei componenti
  - ⇒ Modellazione della struttura e della configurazione
- Il **modello OOA/OOD** viene espresso tramite dei **diagrammi** grafici
- Ogni entità del modello può comparire in uno o più diagrammi, che ne rappresentano una proiezione
- A ogni entità si possono anche associare vari tipi di documentazione testuale
- Nei vari diagrammi, tutti i concetti e le entità che presentano similitudini sono espressi con la medesima notazione

4

# Diagramma vs. modello

In UML c'è distinzione fra i concetti di modello e di diagramma:

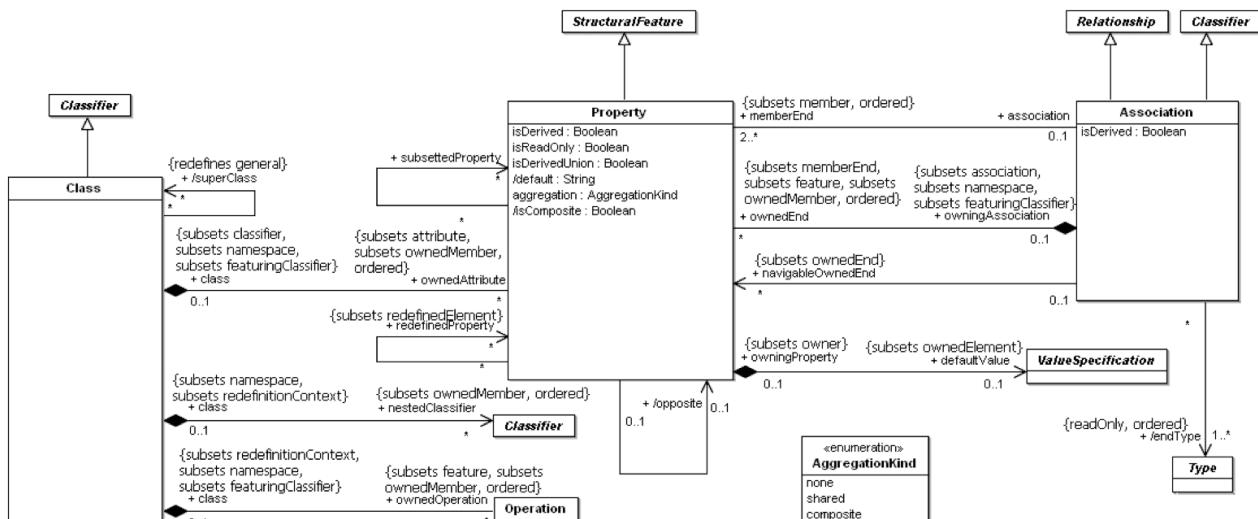
- Un **modello** contiene elementi di informazione circa il sistema sotto osservazione
  - Un **diagramma** è una particolare visualizzazione di alcuni tipi di elementi di un modello

Un certo elemento può comparire in più diagrammi ma è univoca la sua definizione all'interno del modello

5

## Il metamodello di UML

Una piccola porzione del metamodello UML 2 ...



6

## 2

# La struttura di UML

□ La struttura di UML è composta da:

⇒ **costituenti fondamentali**: gli elementi di base

- **entità**
- **relazioni**
- **diagrammi**

⇒ **meccanismi comuni**: tecniche comuni per raggiungere specifici obiettivi

- specifiche
- ornamenti
- distinzioni comuni
- meccanismi di estendibilità

⇒ **architettura**: l'espressione dell'architettura del sistema

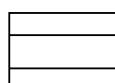
7

## Entità

*Sono gli elementi di modellazione*

### Strutture:

- Classe



- Interfaccia



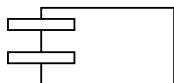
- Collaborazione



- Caso d'uso



- Componente



- Nodo

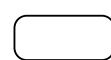


### Comportamenti:

- Interazione

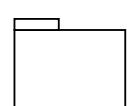


- Stato



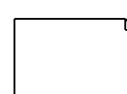
### Raggruppamenti:

- Package



### Informazioni:

- Annotazione



8

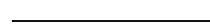
# Relazioni

*Legano tra loro le entità*

- Dipendenza



- Associazione



- Aggregazione



- Contenimento



- Generalizzazione



- Realizzazione



- Composizione



9

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

## Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

10

# Diagrammi

Sono **Statici:** descrive la struttura dati degli oggetti del sistema e le loro relazioni; è il diagramma più importante, da cui si può generare il codice

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite
- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

11

# Diagrammi

Sono viste sul modello UML

**Statici:**

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

mostra un insieme di oggetti di interesse e le loro relazioni

Diagramma dei casi d'uso

- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

12

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

## Dinamici:

- mostra i package e le loro relazioni di dipendenza, contenimento e specializzazione
- a dei casi d'uso
- a degli stati
- Diagramma di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

13

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

## Dinamici:

- descrive l'architettura software del sistema
- a degli stati
- a di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

14

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

## Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
- ✓ Diagramma di sequenza
- ✓ Diagramma di comunicazione
- ✓ Diagramma di sintesi dell'interazione
- ✓ Diagramma dei tempi

describe la struttura del sistema hardware e l'allocazione dei vari moduli software

15

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

## Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
- Diagramma di sequenza
- ✓ Diagramma di comunicazione
- ✓ Diagramma di sintesi dell'interazione
- ✓ Diagramma dei tempi

mostra la struttura interna di classificatori strutturati

16

# Diagrammi

Statici:

- Diagramma delle classi
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

elenca i casi d'uso del sistema e le loro relazioni

UML

Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

17

# Diagrammi

Sono viste sul modello UML

Statici:

- Diagramma degli automi di Harel per descrivere gli stati degli oggetti di una classe
- Diagramma degli oggetti
- Diagramma dei package
- Diagramma dei componenti
- Diagramma di deployment
- Diagramma delle strutture composite

usa la notazione degli automi di Harel per descrivere gli stati degli oggetti di una classe

Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

18

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma dei componenti
- Diagramma dei package
- Diagramma delle strutture composite
- Diagramma di deployment
- Diagramma delle sequenze
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
- ✓ Diagramma di sequenza
- ✓ Diagramma di comunicazione
- ✓ Diagramma di sintesi dell'interazione
- ✓ Diagramma dei tempi

## Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
- ✓ Diagramma di sequenza
- ✓ Diagramma di comunicazione
- ✓ Diagramma di sintesi dell'interazione
- ✓ Diagramma dei tempi

19

# Diagrammi

*Sono viste sul modello UML*

## Statici:

- Diagramma delle classi
- Diagramma dei componenti
- Diagramma dei package
- Diagramma delle strutture composite
- Diagramma di deployment
- Diagramma delle sequenze
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
- ✓ Diagramma di sequenza
- ✓ Diagramma di comunicazione
- ✓ Diagramma di sintesi dell'interazione
- ✓ Diagramma dei tempi

## Dinamici:

- Diagramma dei casi d'uso
- Diagramma degli stati
- Diagramma di attività
- Diagramma di interazione
- ✓ Diagramma di sequenza
- ✓ Diagramma di comunicazione
- ✓ Diagramma di sintesi dell'interazione
- ✓ Diagramma dei tempi

20

# Specifiche

- Sono la descrizione testuale della semantica di un elemento



Caso d'uso: "APRI CONTO CORRENTE BANCARIO"

Scenario base:

- 1 il cliente si presenta in banca per aprire un nuovo c/c
- 2 l'addetto riceve il cliente e fornisce spiegazioni
- 3 se il cliente accetta fornisce i propri dati
- 4 l'addetto verifica se il cliente è censito in anagrafica
- 5 l'addetto crea il nuovo conto corrente
- 6 l'addetto segnala il numero di conto al cliente

Varianti:

- 3(a) se il cliente non accetta il caso d'uso termina
- 3(b) se il conto va intestato a più persone vanno forniti i dati di tutte
- 4(a) se il cliente (o uno dei diversi intestatari) non è censito l'addetto provvede a registrarlo, richiede al cliente la firma dello specimen e ne effettua la memorizzazione via scanner

21

# Ornamenti

- Rendono visibili gli aspetti particolari della specifica dell'elemento



Finestra  
{autore = Smith}

+dimensioni: Rettangolo=(100,100)  
#visible: Booleano=falso  
+dimensioniPredefinite: Rettangolo

+crea()  
+nascondi()

22

# Distinzioni comuni

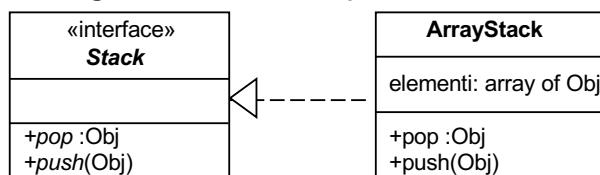
## □ Classificatore/istanza

- ⇒ Separa la nozione astratta di un'entità dalle sue concrete istanze
- ⇒ Un'istanza ha di solito la stessa forma del classificatore, ma con il nome sottolineato



## □ Interfaccia/implementazione

- ⇒ Separa “cosa” un oggetto fa da “come” lo fa
- ⇒ Un’interfaccia definisce un contratto che ciascuna sua implementazione garantisce di rispettare

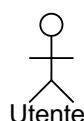


23

# Meccanismi di estendibilità

- Uno **stereotipo** rappresenta una variazione di un elemento di modellazione esistente, con la stessa forma ma diverso scopo. Permette quindi di introdurre nuovi elementi di modellazione a partire da quelli esistenti
  - ⇒ predefiniti
  - ⇒ introdotti dall’utente

Attore è stereotipo  
(con parentesi angolari <>>)



- Una **proprietà** è un valore associato a un elemento del modello, espresso da una stringa associata all’elemento

```
{ author = "Joe Smith", status = analysis } { abstract }
```
- Un **vincolo** è una frase di testo che definisce una condizione o una regola che riguarda un elemento del modello e deve risultare sempre vera

```
{ disjoint, complete } { subset }
```
- Un **profilo** è un insieme di stereotipi, valori etichettati (che definiscono proprietà) e vincoli, usato per personalizzare UML

24

# Architettura

- **Vista dei casi d'uso**
  - ⇒ Descrive le funzionalità del sistema come vengono percepite dagli utenti, dagli analisti e dagli esecutori del testing. Non specifica l'organizzazione del software ma è la base per le altre viste
- **Vista logica**
  - ⇒ Stabilisce la terminologia del dominio del problema sotto forma di classi e oggetti, illustrando come essi implementano il comportamento richiesto
- **Vista dei processi**
  - ⇒ È una variante orientata ai processi della vista logica; modella i thread e i processi sotto forma di classi attive
- **Vista di implementazione**
  - ⇒ Descrive i moduli implementativi e le loro dipendenze, illustrandone la configurazione così da definire il concetto di versione del sistema
- **Vista di deployment**
  - ⇒ Mostra la distribuzione fisica del sistema software sull'architettura hardware

25

## Viste/diagrammi (sistema complesso)

	casi d'uso	logica	dei processi	implementativa	di deployment
casi d'uso	X				
classi/oggetti		X	X		
componenti				X	
distribuzione					X
stato		X	X	X	X
attività	X	X	X	X	X
interazione	X	X	X	X	X

aspetti statici - aspetti dinamici

26

## Viste/diagrammi (sistema medio)

	casi d'uso	logica	dei processi	implementativa	di deployment
casi d'uso	X				
classi/oggetti		X	X		
componenti				X	
distribuzione					X
stato		X			
attività	X				
interazione		X	X		

aspetti statici - aspetti dinamici

27

## Viste/diagrammi (sistema piccolo)

	casi d'uso	logica	dei processi	implementativa	di deployment
casi d'uso	X				
classi/oggetti		X			
componenti				(X)	
distribuzione					(X)
stato		(X)			
attività					
interazione		X			

aspetti statici - aspetti dinamici

28

# 3

# Diagrammi dei casi d'uso

- Rappresentano i *ruoli* di utilizzo del sistema da parte di uno o più utilizzatori (**attori**):
  - ⇒ esseri umani (dipendenti, clienti)
  - ⇒ organizzazioni, enti, istituzioni
  - ⇒ altre applicazioni o sistemi (hardware e software), sottosistemi
- Descrivono l'**interazione** tra attori e sistema, non la logica interna della funzione né la struttura del sistema
- Sono espressi in forma **testuale**, comprensibile anche per i non "addetti ai lavori"
- Possono essere definiti a livelli diversi (l'intero sistema o parti del sistema), ma sempre dal punto di vista dell'utente

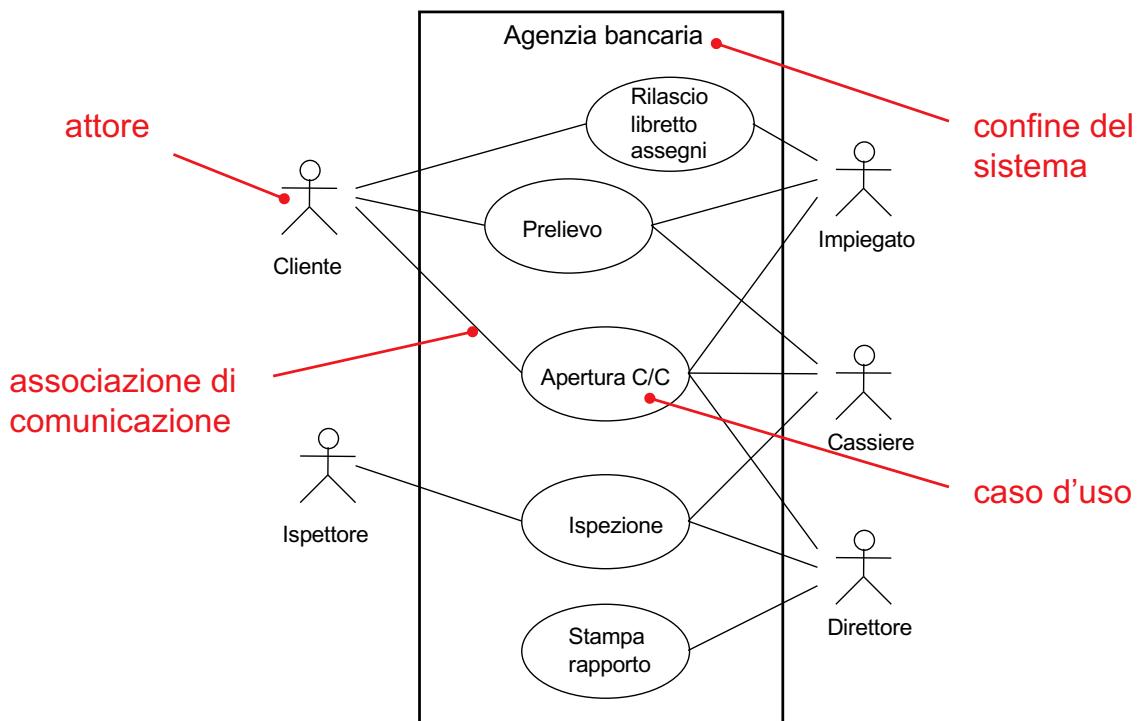
29

## Attore vs. caso d'uso

- Un **attore** identifica il ruolo che un'entità esterna assume quando interagisce direttamente con il sistema
  - ⇒ ... è sempre esterno al sistema, anche se il sistema ne può mantenere una rappresentazione interna
  - ⇒ ... spedisce o riceve messaggi dal sistema, o scambia informazioni con esso
  - ⇒ ... esegue i casi d'uso
  - ⇒ ... è modellato con una classe, non un oggetto
- Un **caso d'uso** è la specifica di una sequenza di azioni che un sistema, un sottosistema o una classe può eseguire interagendo con attori esterni
  - ⇒ ... è una funzionalità come percepita da un attore
  - ⇒ ... produce un risultato osservabile utile all'attore
  - ⇒ ... viene sempre attivato da un attore
  - ⇒ ... è completo

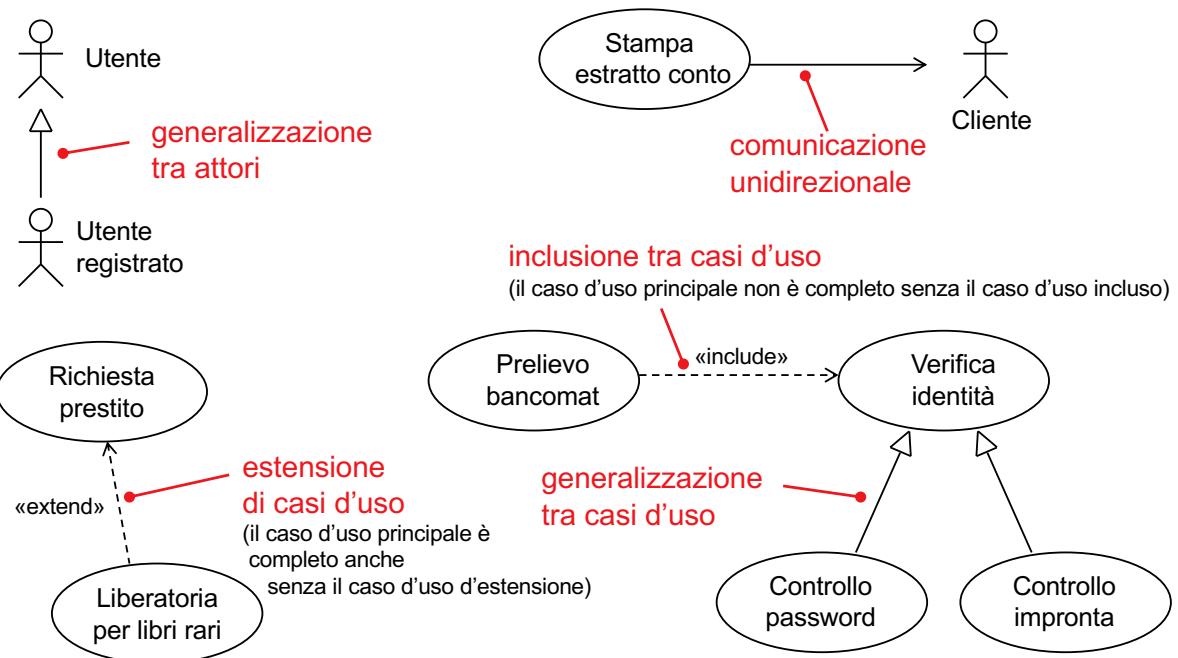
30

# I casi d'uso di una banca



31

## Relazioni nei diagrammi dei casi d'uso



32

# Punti di vista



## UTILIZZATORE

- Casi d'uso
  - ⇒ telefonare
  - ⇒ ricevere telefonate
  - ⇒ inviare messaggi
  - ⇒ memorizzare un numero
  - ⇒ ....

## PROGETTISTA

- Funzionalità interne
  - ⇒ trasmissione / ricezione
  - ⇒ alimentazione (batteria)
  - ⇒ I/O (display, tasti, ...)
  - ⇒ gestione rubrica
  - ⇒ .....

33

# Ruolo dei casi d'uso

- Nelle fasi iniziali della progettazione servono per chiarire cosa dovrà fare il sistema
  - ⇒ Ragionare sui casi d'uso con il committente è uno dei modi più efficaci ed efficienti per scoprire ed analizzare i requisiti ai quali il sistema dovrà fornire un'implementazione
  - ⇒ Dialogare su come il sistema verrà utilizzato, nella comunicazione con persone non esperte nella progettazione, è certamente più facile che non guardare a come dovrà essere costruito
  - ⇒ Raggiungere un accordo con il committente sulle modalità di utilizzo del sistema consente al progettista di affrontare con maggiore tranquillità il suo mestiere specifico di progettazione
- I casi d'uso guidano l'intero progetto di sviluppo
  - ⇒ Costituiscono il punto di partenza per la progettazione del sistema
  - ⇒ Sono il riferimento primario per la definizione, la progettazione, l'esecuzione dei test per la verifica di quanto prodotto
  - ⇒ Rappresentano delle naturali unità di rilascio, per i progetti che seguono un approccio incrementale alla pianificazione della realizzazione e dei rilasci

34

# Identificare i casi d'uso

1. Individuare i confini del sistema
2. Identificare tutte le tipologie di utilizzatori del sistema (esseri umani o altri sistemi), che verranno modellati come attori
3. Per ogni tipologia di attore, rilevare in quale modo utilizzerà il sistema, partendo dagli obiettivi che egli deve raggiungere. A ogni modalità di utilizzo corrisponde un caso d'uso
4. Per ogni caso d'uso, descrivere lo scenario base (la sequenza di passi più semplice possibile che conduce al successo del caso d'uso, le risposte attese dal sistema), e le principali varianti a tale scenario. Così facendo, tipicamente, possono emergere necessità di interazione del sistema con altri soggetti (esseri umani o altri sistemi), che verranno rappresentati nel modello come attori aggiuntivi

35

# Scenari

- Ogni specifica esecuzione (istanza) di un caso d'uso è detta **scenario**
  - ⇒ Ad esempio, in un caso d'uso "acquisto di un prodotto", ogni specifico acquisto effettuato da uno specifico cliente in uno specifico momento costituisce uno scenario particolare
- Esistono scenari di **successo** e scenari di **fallimento**
- Gli scenari possibili sono innumerevoli
- La prassi più diffusa per la descrizione degli scenari di un caso d'uso è quella di definire uno **scenario base**, cioè lo scenario più semplice possibile che porta al successo del caso d'uso
- Allo scenario base vengono quindi agganciate le **varianti**, che lo rendono più complesso e possono portare al successo o al fallimento del caso d'uso

36

# Scenari

Caso d'uso: "APRI CONTO CORRENTE BANCARIO"

Scenario base:

- 1 il cliente si presenta in banca per aprire un nuovo c/c
- 2 l'addetto riceve il cliente e fornisce spiegazioni
- 3 se il cliente accetta fornisce i propri dati
- 4 l'addetto verifica se il cliente è censito in anagrafica
- 5 l'addetto crea il nuovo conto corrente
- 6 l'addetto segnala il numero di conto al cliente

Varianti:

- 3(a) se il cliente non accetta il caso d'uso termina
- 3(b) se il conto va intestato a più persone vanno forniti i dati di tutte
- 4(a) se il cliente (o uno dei diversi intestatari) non è censito l'addetto provvede a registrarlo, richiede al cliente la firma dello specimen e ne effettua la memorizzazione via scanner

37

## Specifiche del caso d'uso

- UML non suggerisce il modo per specificare un caso d'uso, lasciando spazio libero a tutte le possibili forme di documentazione testuale
- La specifica del caso d'uso, comunque effettuata, ha un ruolo centrale nella comunicazione tra i diversi soggetti coinvolti nello sviluppo di un sistema, dal committente agli utilizzatori, dai progettisti agli specialisti di test
- Un caso d'uso può essere anche descritto da un **diagramma di attività** o **di sequenza**

38

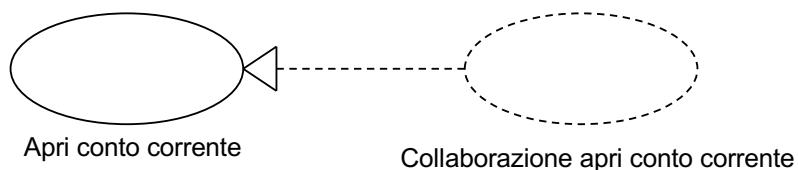
# Specifiche del caso d'uso

Nome	
Identificatore	
Breve descrizione	<i>fissa l'obiettivo del caso d'uso</i>
Attori primari	<i>avviano il caso d'uso</i>
Attori secondari	<i>interagiscono con il caso d'uso dopo che è stato avviato</i>
Precondizioni	<i>condizioni che devono essere vere prima che il caso d'uso possa essere eseguito</i>
Sequenza principale degli eventi	<i>i passi che costituiscono il caso d'uso</i>
Postcondizioni	<i>condizioni che devono essere vere quando il caso d'uso termina</i>
Sequenze alternative degli eventi	<i>un elenco di alternative alla sequenza principale</i>

39

# Realizzare i casi d'uso

- La realizzazione dei casi d'uso può essere espressa con una **collaborazione** costituita da classi che interagendo tra loro svolgono i passi specificati nel caso d'uso
- La collaborazione che realizza un caso d'uso può essere descritta:
  - ⇒ a livello **statico** mediante un diagramma delle classi che evidenzia le classi o gli oggetti coinvolti nella collaborazione
  - ⇒ a livello **dinamico** mediante un diagramma di interazione che evidenzia i messaggi che gli oggetti si scambiano nell'ambito della collaborazione



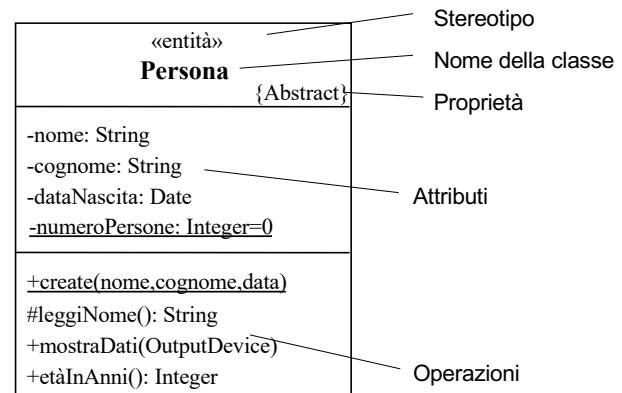
40

# 4

# Diagrammi delle classi

- Sono il nucleo fondamentale di UML
- Descrivono la struttura statica del sistema in termini di classi e loro relazioni reciproche

- ⇒ Una **classe** descrive un gruppo di oggetti con proprietà, comportamento e relazioni comuni
- ⇒ Un **attributo** è un valore che caratterizza gli oggetti di una classe
- ⇒ Un'**operazione** è una trasformazione che può essere applicata a (o invocata da) gli oggetti di una classe. Ogni operazione ha come argomento implicito l'oggetto destinazione



41

## Notazione

- Per gli **attributi** della classe:

visibilità nome molteplicità : tipo = valoreDefault

- ⇒ Visibilità
  - pubblica +
  - privata -
  - protetta #
  - package ~
- ⇒ Molteplicità
  - per esempio: String [5], Real [2..\*], Boolean [0..1]
- ⇒ Tipo
  - Integer, UnlimitedNatural, Real
  - Boolean
  - String
- ⇒ Ambito
  - istanza
  - classe

42

# Notazione

## □ Per le **operazioni** della classe:

visibilità nome (parametro, ...): tipoRestituito

*signature*

### ⇒ Parametri

direzione nomeParametro: tipoParametro=valoreDefault

### ⇒ Direzione

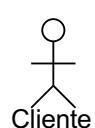
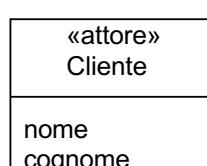
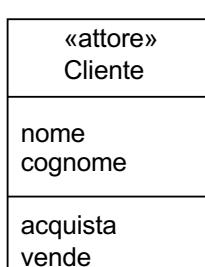
- in
- out
- inout
- return (si usa quando l'operazione restituisce più valori)

### ⇒ Ambito

- istanza
- classe

43

# Diversi livelli di astrazione



44

# Le relazioni tra classi

- Generalizzazione →
- Associazione ——————
- Dipendenza -----→
- Aggregazione ——————◊
- Composizione ——————◆
- Raffinamento -----→

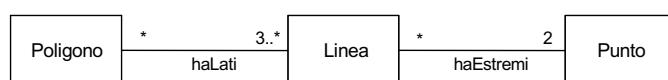
45

## Associazione

- E' una connessione tra classi, tipicamente bidirezionale
- Molteplicità:

- ⇒ Esattamente 1
- ⇒ Opzionale 1
- ⇒ Da x a y inclusi
- ⇒ Solo i valori a,b,c
- ⇒ 1 o più
- ⇒ 0 o più

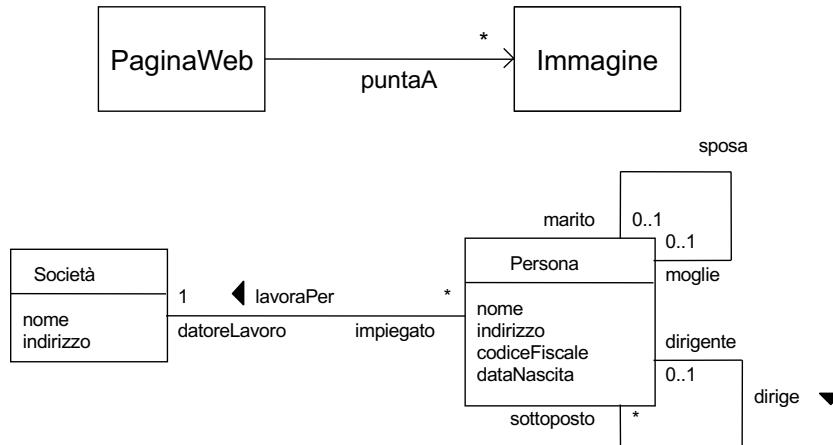
1
0..1
x..y
a,b,c
1..*
*



46

# Associazione

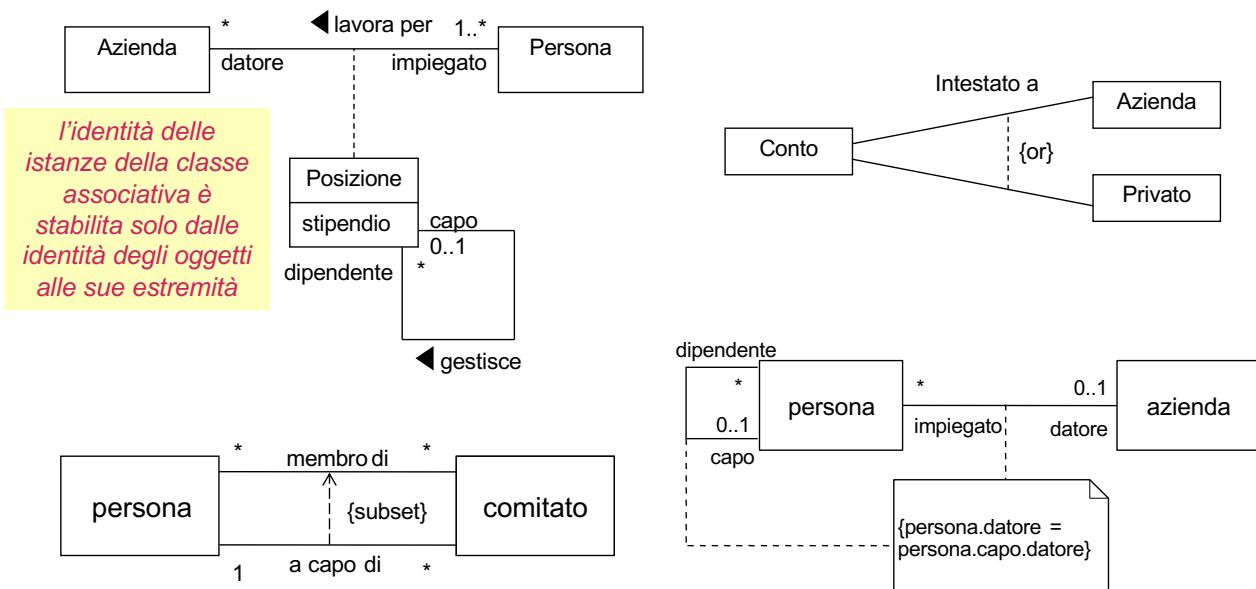
- E' possibile indicare il **verso di lettura** di una associazione, definire associazioni **monodirezionali**, specificare **ruoli**



47

# Associazione

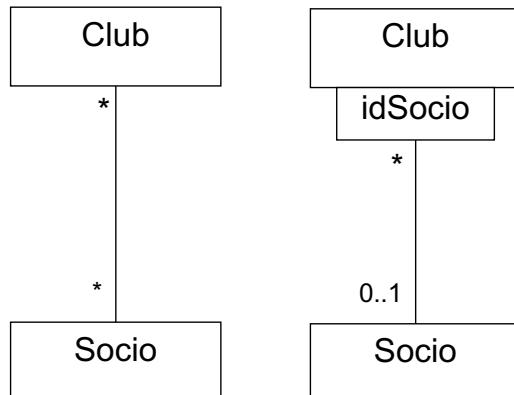
- E' possibile specificare **vincoli** e **classi associative**



48

# Associazione

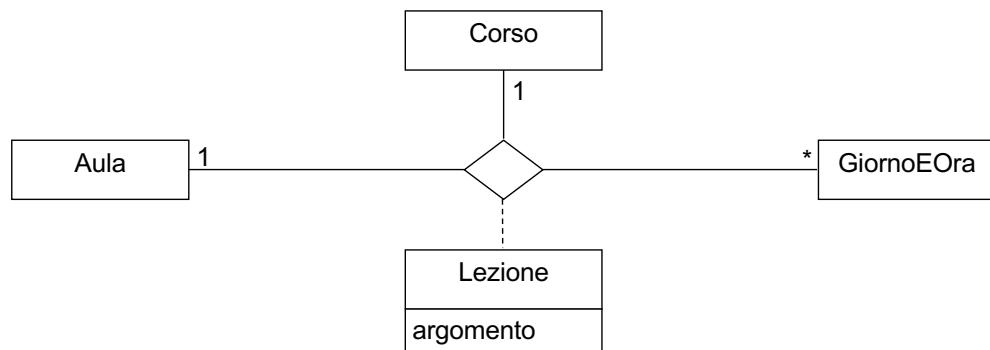
- Le **associazioni qualificate** riducono un'associazione molti-a-molti a una del tipo uno-a-uno, specificando un attributo che permette di selezionare un unico oggetto destinazione svolgendo il ruolo di identificatore o chiave di ricerca



49

# Associazione

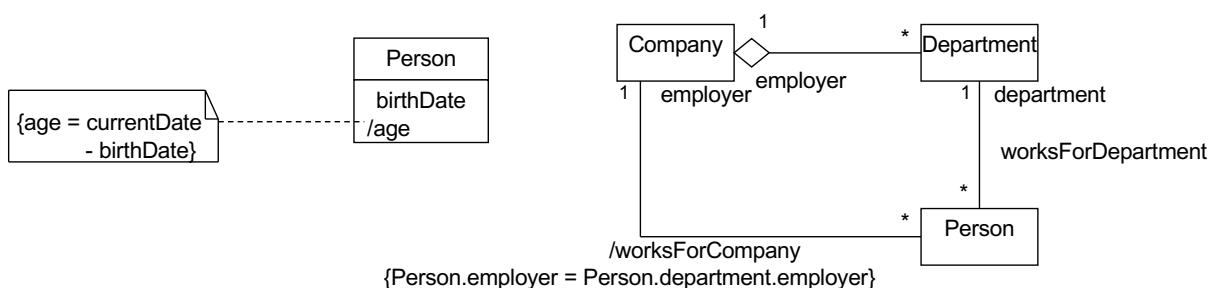
- E' possibile definire **associazioni n-arie** (cioè tra n classi)
  - Ogni istanza dell'associazione è una tupla formata da n oggetti delle rispettive classi
  - La molteplicità di un ruolo rappresenta il numero di istanze dell'associazione quando sono stati fissati n-1 oggetti
  - I numeri di istanze dell'associazione quando è fissato un solo oggetto sono implicitamente assunti essere tutti a "molti"



50

## Elementi derivati

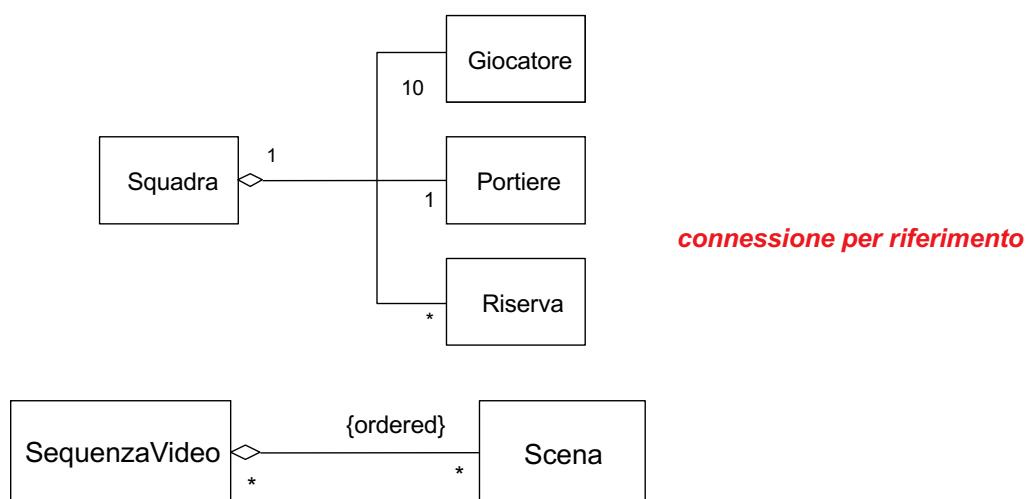
- Un **elemento derivato** può essere calcolato a partire da un altro ma viene mostrato, per motivi di chiarezza o per scelte di progettazione, nonostante non aggiunga alcuna ulteriore informazione semantica
  - ⇒ viene indicato posizionando uno slash prima del suo nome
  - ⇒ i dettagli su come calcolarlo possono essere inseriti in una nota o essere rappresentati con una stringa di vincoli



51

## Aggregazione

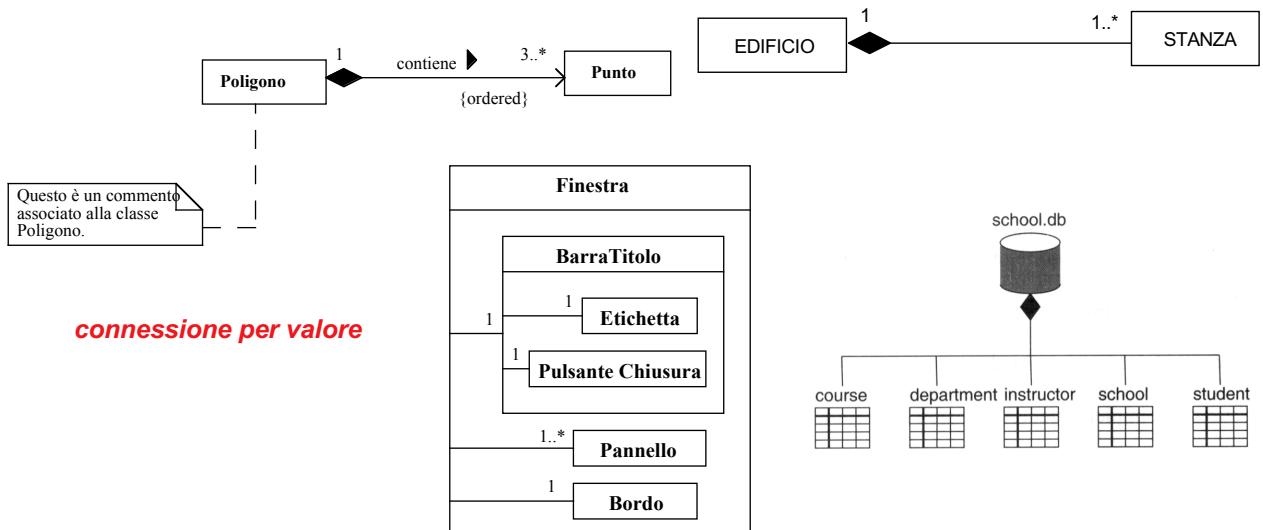
- E' un caso speciale di associazione con semantica *part-of*
  - ⇒ Sia il tutto che le parti esistono indipendentemente



52

# Composizione

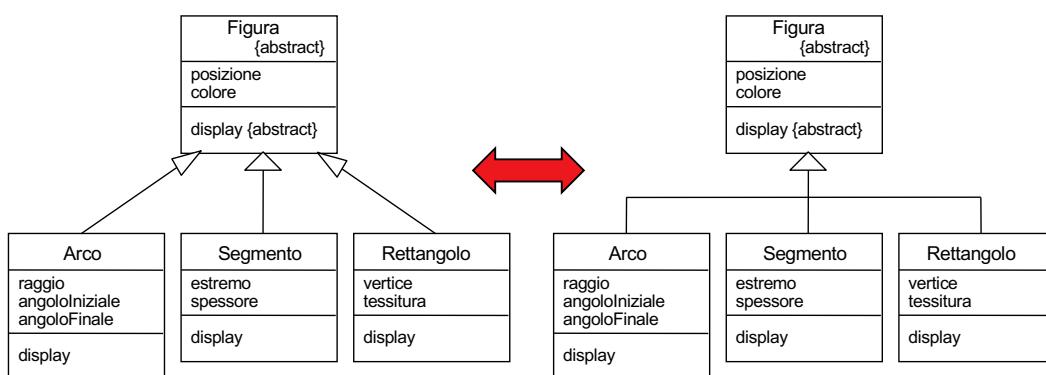
- E' un'aggregazione in cui il tutto "possiede" le sue parti
  - ⇒ Le parti esistono solo in relazione al tutto
  - ⇒ Ogni parte appartiene a esattamente un tutto



53

# Generalizzazione

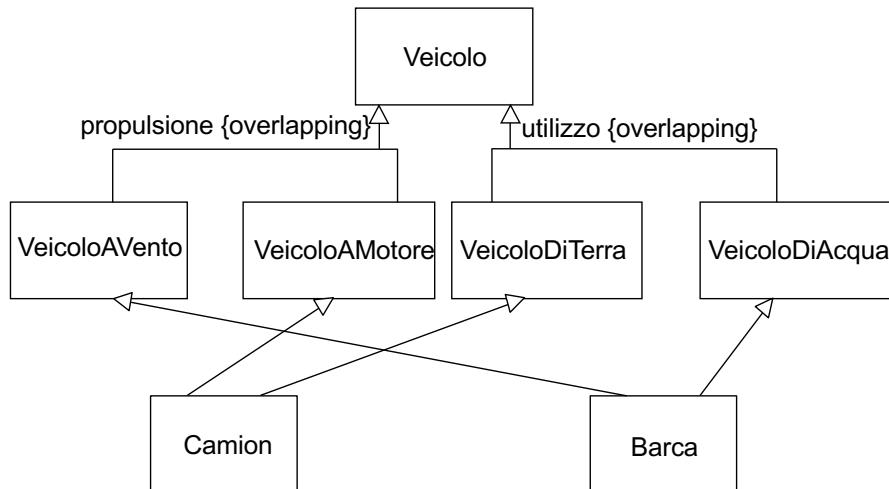
- Tutti gli attributi, le operazioni e le relazioni della superclasse vengono **ereditati** dalle sottoclassi



54

# Generalizzazione

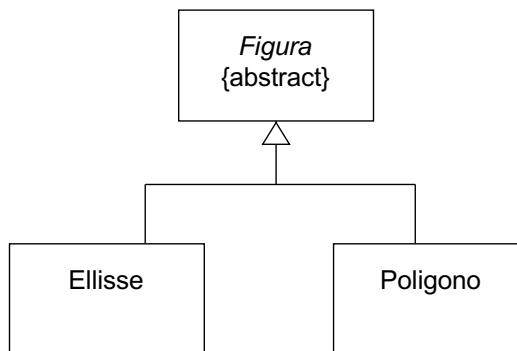
- E' supportata l'**ereditarietà multipla**
- Possono essere indicati **insiemi di generalizzazione** e vincoli (*overlapping, disjoint, complete, incomplete*)



55

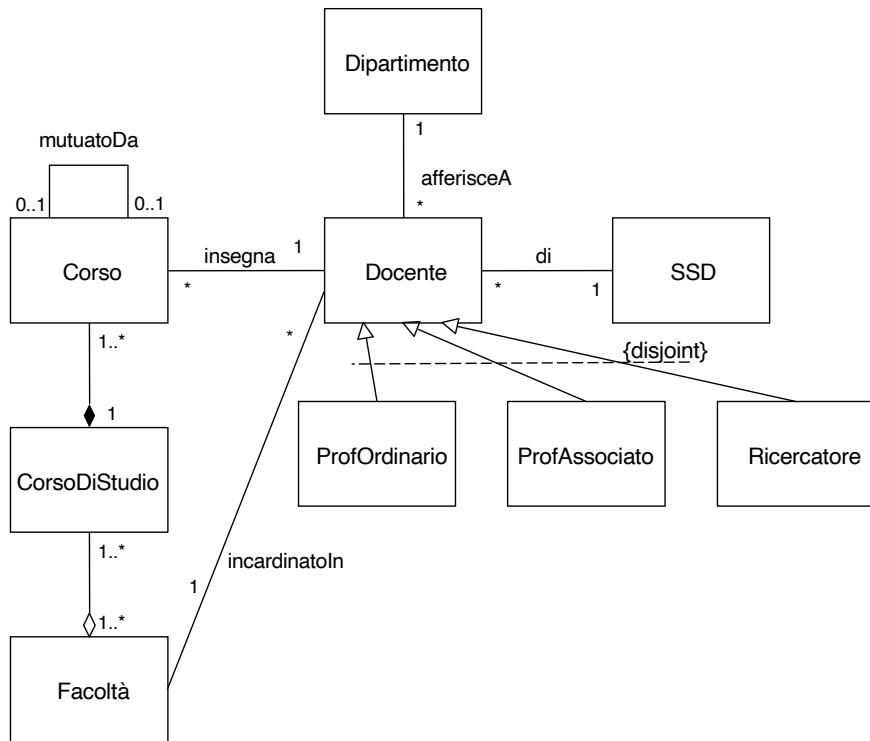
# Classi astratte

- Sono classi che non possono essere istanziate da oggetti
- Sono utili come radici di **gerarchie di specializzazione**



56

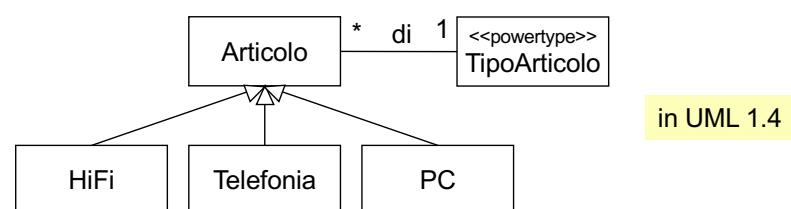
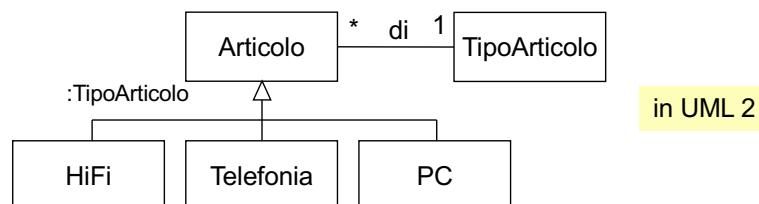
# Un esempio



57

# Powertyping

- Un *powertype* è una (meta)classe le cui istanze sono classi che specializzano un'altra classe



58

# Dipendenza

- In generale, *A dipende da B quando una variazione in B può comportare una variazione in A*
- Nel caso delle classi, una dipendenza indica che una classe cliente dipende da alcuni servizi di una classe fornitrice, ma non ha una struttura interna che dipende da quest'ultima
  - ⇒ Lo stereotipo più comunemente usato è «use»



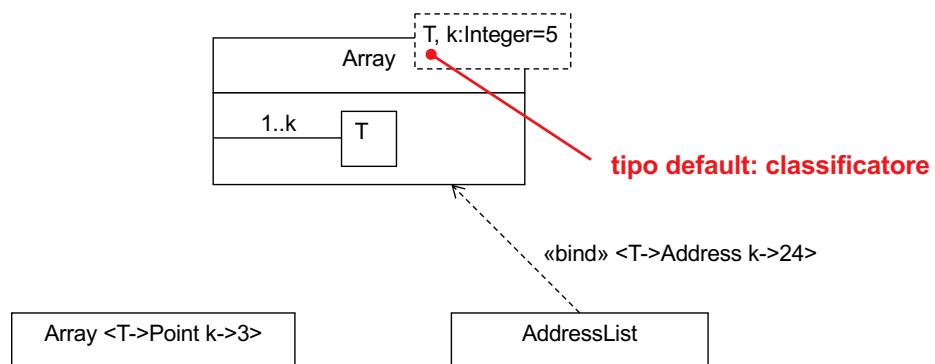
- ⇒ Più specificamente, si può rappresentare il fatto che un'operazione della classe cliente ha argomenti che appartengono al tipo di un'altra classe



59

# Template

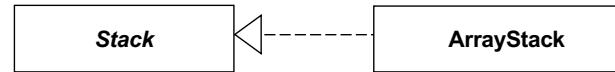
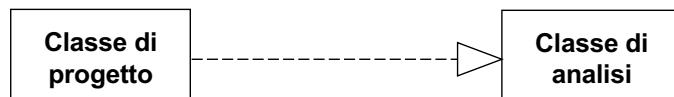
- Un *template* (o *classe parametrizzata*) è utilizzato per descrivere una classe in cui uno o più parametri formali non sono istanziati
  - ⇒ Un template definisce una famiglia di classi in cui ogni classe è specificata istanziando i parametri con i valori attuali
  - ⇒ Un template non è utilizzabile direttamente
- Un *bound element* è una classe che istanzia i parametri di un template, e può essere utilizzato esattamente come una classe



60

# Raffinamento

- Esprime una relazione tra due descrizioni dello stesso concetto a diversi livelli di astrazione
  - ⇒ Tra un tipo astratto e una classe che lo realizza (*realizzazione*)
  - ⇒ Tra una classe di analisi e una di progetto
  - ⇒ Tra una implementazione semplice e una complessa della stessa cosa



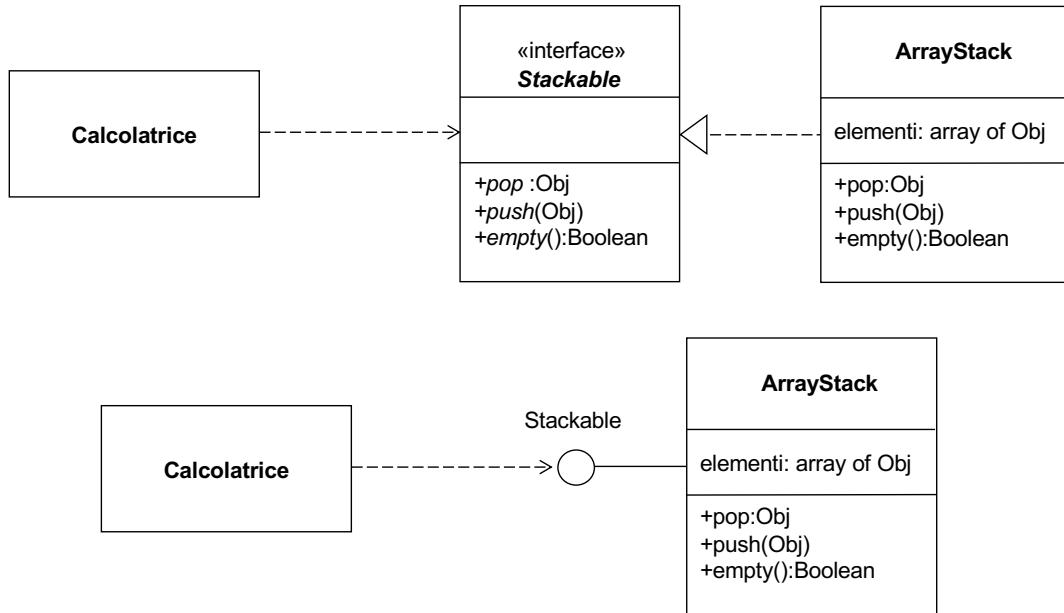
61

# Interfaccia

- Una *interfaccia* è un insieme di funzionalità pubbliche identificate da un nome
- Specifica le operazioni pubbliche di una classe, di un componente, di un pacchetto o di altre entità, separandone le specifiche dall'implementazione
- Un'interfaccia non ha alcuna specifica di struttura interna (attributi, stato o associazioni); è una classe astratta, senza attributi né associazioni e con solo operazioni astratte (senza implementazione)
  - ⇒ La notazione estesa prevede una rappresentazione simile a quella delle classi, con «interface» come stereotipo e senza comportamento per gli attributi; la notazione minimizzata prevede un piccolo cerchio collegato all'entità (classe, componente o package) che la supporta, col nome dell'interfaccia vicino (*lollipop notation*)
  - ⇒ Un'altra classe che usa l'interfaccia può essere collegata ad essa da una freccia di dipendenza, eventualmente con lo stereotipo «use»

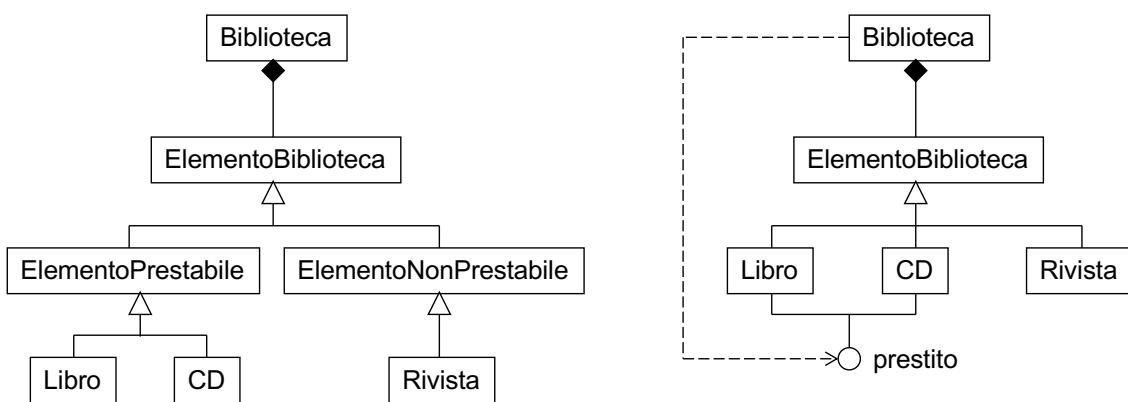
62

# Interfaccia



63

# Interfaccia vs. ereditarietà



64

# Analisi vs. progettazione

## □ Classi di analisi

- ⇒ rappresentano un'astrazione nel dominio del problema
- ⇒ corrispondono chiaramente a concetti concreti del mondo del business
- ⇒ escludono tutti i dettagli implementativi
- ⇒ hanno un insieme ridotto, coeso e ben definito di responsabilità
- ⇒ indicano gli attributi che saranno *probabilmente* inclusi nelle classi di progettazione
- ⇒ le loro operazioni specificano i principali servizi offerti dalla classe

## □ Classi di progettazione

- ⇒ le loro specifiche sono complete per cui possono essere direttamente implementate
- ⇒ nascono dal dominio del problema per raffinamento delle classi di analisi, oppure dal dominio della soluzione

65

## Identificare le classi d'analisi

- ### □ Le classi corrispondono a **entità fisiche** e a **concetti** del dominio applicativo
- ⇒ Evitare di rappresentare soluzioni implementative



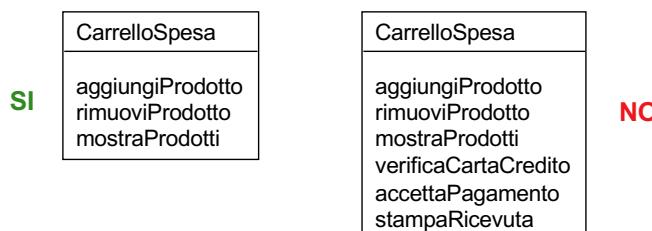
- ⇒ Evitare le classi ridondanti, irrilevanti, vaghe
- ⇒ Evitare le classi “onnipotenti”



66

## Identificare le classi d'analisi

- ⇒ Una classe è associata a un piccolo e ben definito insieme di responsabilità (normalmente tra 3 e 5)

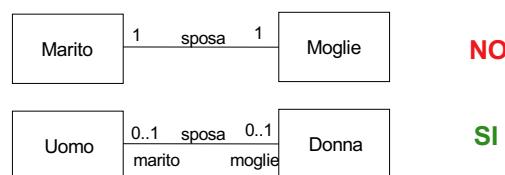


- ⇒ Nessuna classe può essere isolata
- ⇒ Evitare di avere poche classi troppo complesse, ma anche tante classi troppo semplici

67

## Identificare le classi d'analisi

- ⇒ I nomi delle classi devono riflettere la loro natura intrinseca e non il ruolo giocato nelle associazioni

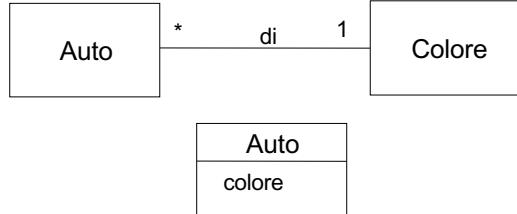


- ⇒ Evitare le gerarchie di specializzazione profonde

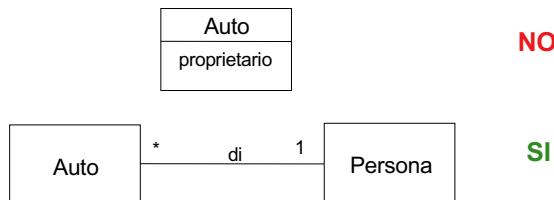
68

## Identificare le classi d'analisi

- ⇒ I nomi che descrivono oggetti dovrebbero essere espressi come attributi



- ⇒ Se una proprietà esiste indipendentemente, o compare più volte all'interno del diagramma, dovrebbe essere espressa come classe

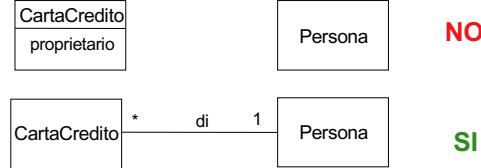


69

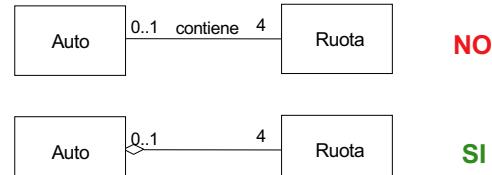
## Identificare le associazioni d'analisi

- Le associazioni sono tipicamente indicate da **verbi** che esprimono collocazione fisica (**contenuto in**), azioni (**gestisce**), comunicazioni (**parla a**), proprietà (**possiede**), soddisfacimento di condizioni (**sposato a**)

- ⇒ Ogni riferimento da una classe a un'altra è un'associazione



- ⇒ Un'aggregazione è un'associazione con semantica *part-of*

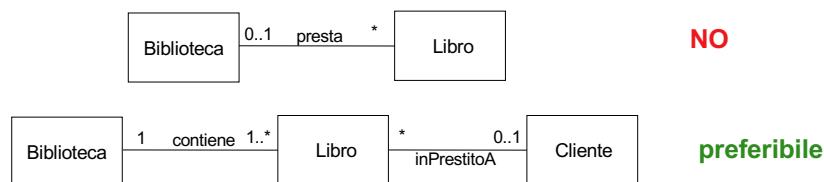


- ⇒ Evitare le associazioni irrilevanti o che esprimono soluzioni implementative

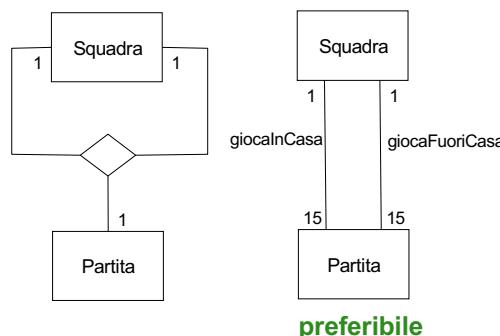
70

# Identificare le associazioni d'analisi

- ➡ Un'associazione deve descrivere una proprietà strutturale del dominio, non un evento transitorio



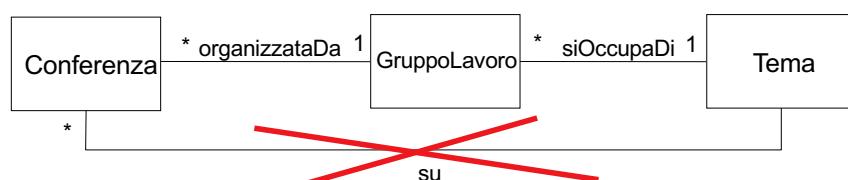
- ➡ Molte associazioni ternarie possono essere scomposte in due associazioni binarie



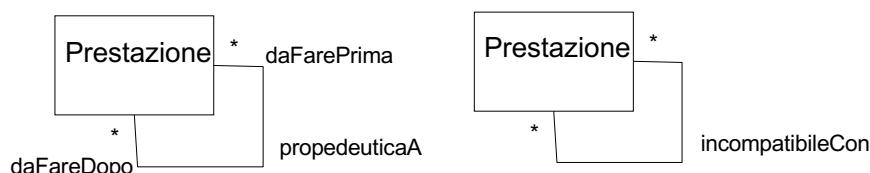
71

# Identificare le associazioni d'analisi

- ➡ Evidenziare le associazioni derivate, che cioè possono essere espresse in termini di altre associazioni



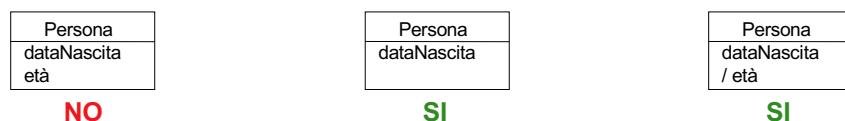
- ➡ Quando appropriato, specificare i ruoli



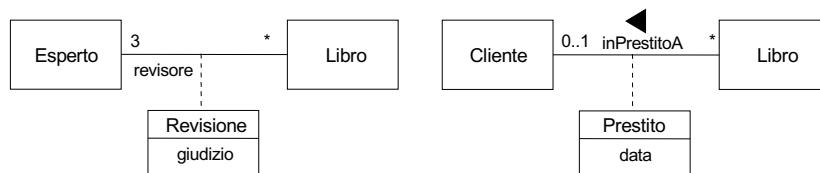
72

## Identificare gli attributi

- Le proprietà di classi e associazioni sono attributi
  - Gli attributi spesso corrispondono a nomi seguiti da possessivi (ad esempio, *il colore della macchina*)
    - ➡ Omettere o evidenziare gli attributi derivati



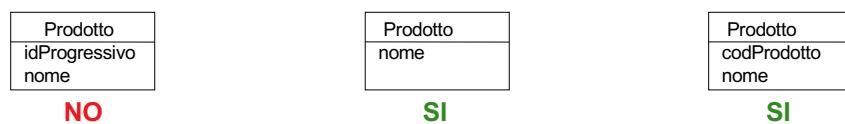
- ➡ Se una proprietà dipende dalla presenza di un'associazione, rappresentarla con un attributo dell'associazione



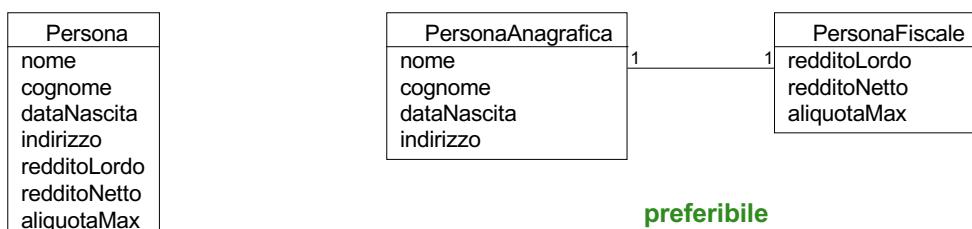
73

## Identificare gli attributi

- ➡ Non aggiungere agli attributi gli identificatori degli oggetti, a meno che non risultino esplicitamente dalle specifiche



- Quando gli attributi di una classe possono essere raggruppati in due o più insiemi, probabilmente la classe dovrebbe essere suddivisa in due o più classi

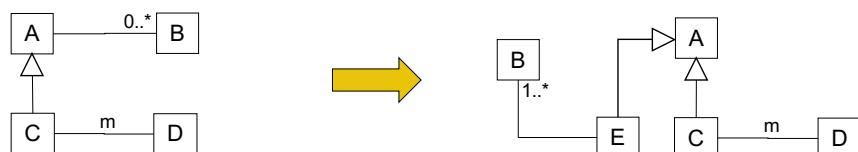


## **preferibile**

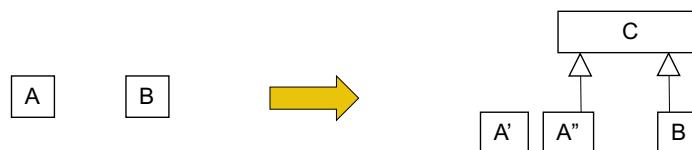
74

# Raffinamenti

- La possibilità di raffinare il modello deriva dalla natura iterativa dell'approccio a oggetti
- I raffinamenti tramite ereditarietà possono avvenire top-down (definizione di specializzazioni di classi esistenti) o bottom-up (generalizzazione di due o più classi con caratteristiche comuni)
  - ↳ Valutare l'utilità di aggiungere nuove classi in caso di asimmetrie in associazioni o generalizzazioni



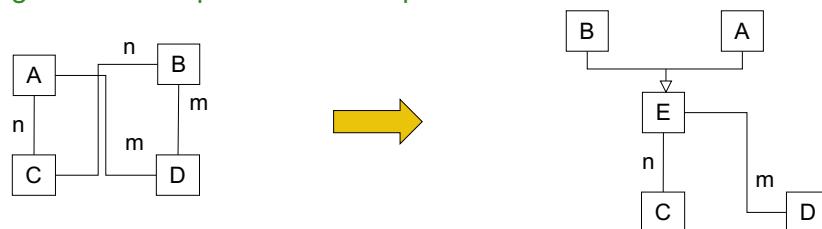
- ↳ In caso di difficoltà nel generalizzare, forse una classe sta giocando due ruoli differenti: può convenire spezzarla in due classi



75

# Raffinamenti

- ↳ Se esistono più associazioni con lo stesso nome e scopo, conviene generalizzare per creare la superclasse che le unisce



- ↳ Se un ruolo incide sostanzialmente sulla semantica della classe, può convenire trasformarlo in una nuova classe



- ↳ Una classe senza attributi, né operazioni, né associazioni può essere eliminata
- ↳ Se nessuna operazione usa un'associazione, forse quella associazione è inutile

76

## Identificare le classi di progettazione

- Con le classi di progettazione si specifica esattamente **come** le classi assolveranno le loro responsabilità
- Ciascuna classe deve essere:
  - ⇒ **completa**, ossia fornire ai suoi clienti tutti i servizi che essi si aspettano
  - ⇒ **sufficiente**, ossia i suoi metodi devono essere esclusivamente finalizzati allo scopo della classe
  - ⇒ **essenziale**, ossia non mettere a disposizione più di un modo per effettuare la stessa operazione
  - ⇒ **massimamente coesa**, ossia modellare un unico concetto astratto
  - ⇒ **minimamente interdipendente**, ossia essere associata all'insieme minimo di classi che consente di realizzare le proprie responsabilità

77

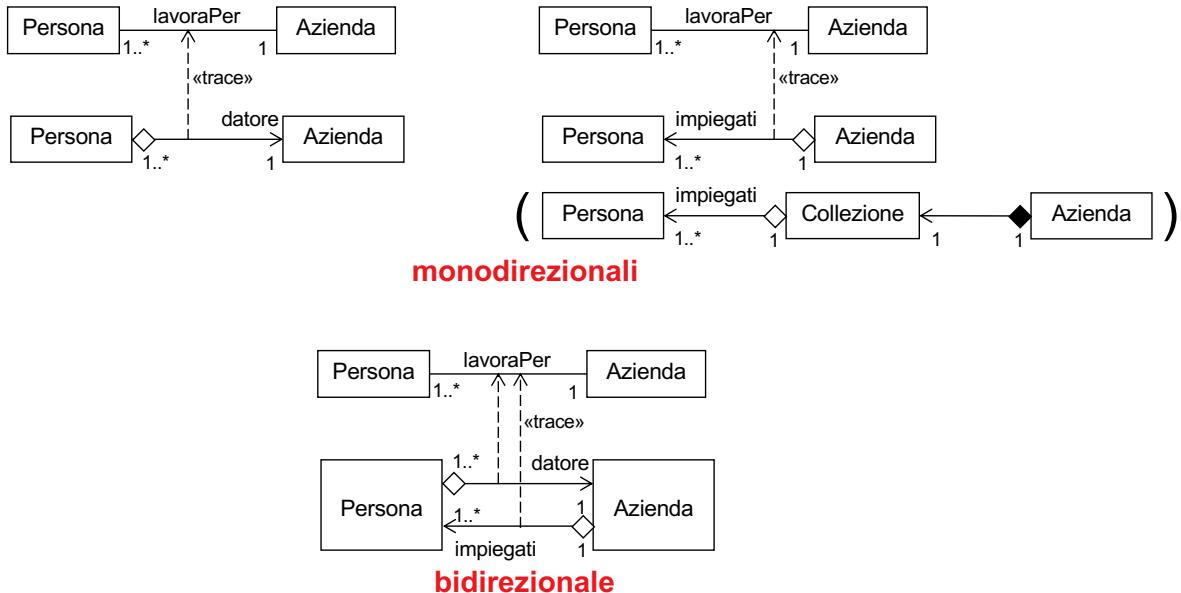
## Identificare le associazioni di progettazione

- Costrutti come le associazioni bidirezionali o le classi associative non sono direttamente implementabili
- Le associazioni di progettazione si ottengono da quelle di analisi attraverso una trasformazione basata principalmente sul carico di lavoro cui ciascuna associazione è sottoposta
- Le associazioni di progettazione *devono* specificare:
  - ⇒ il nome
  - ⇒ il verso di navigabilità
  - ⇒ la molteplicità a entrambi gli estremi
  - ⇒ il nome del ruolo destinazione

78

# Identificare le associazioni di progettazione

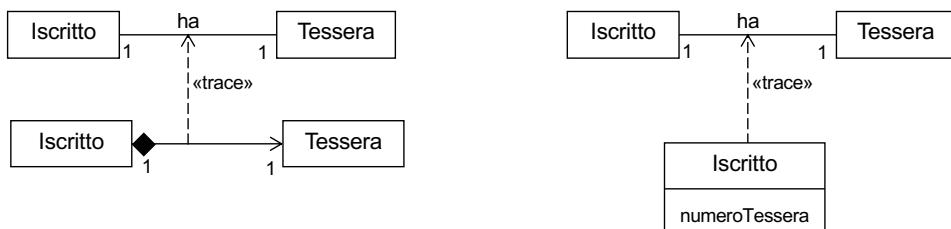
## □ Associazioni multi-a-uno o multi-a-molti



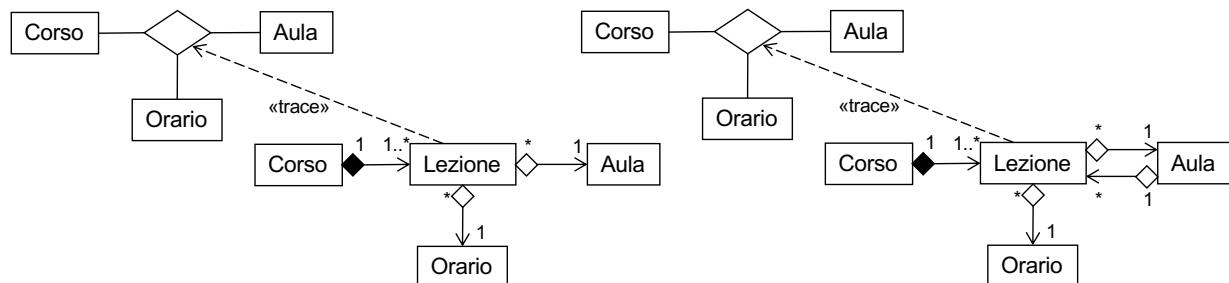
79

# Identificare le associazioni di progettazione

## □ Associazioni uno-a-uno



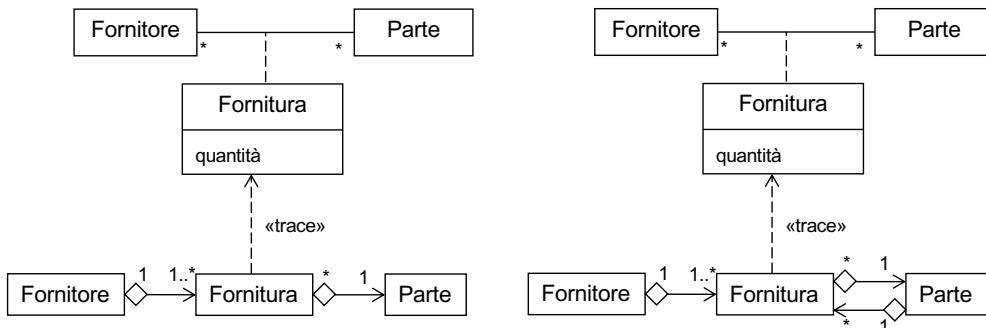
## □ Associazioni ternarie



80

# Identificare le associazioni di progettazione

## □ Classi associative

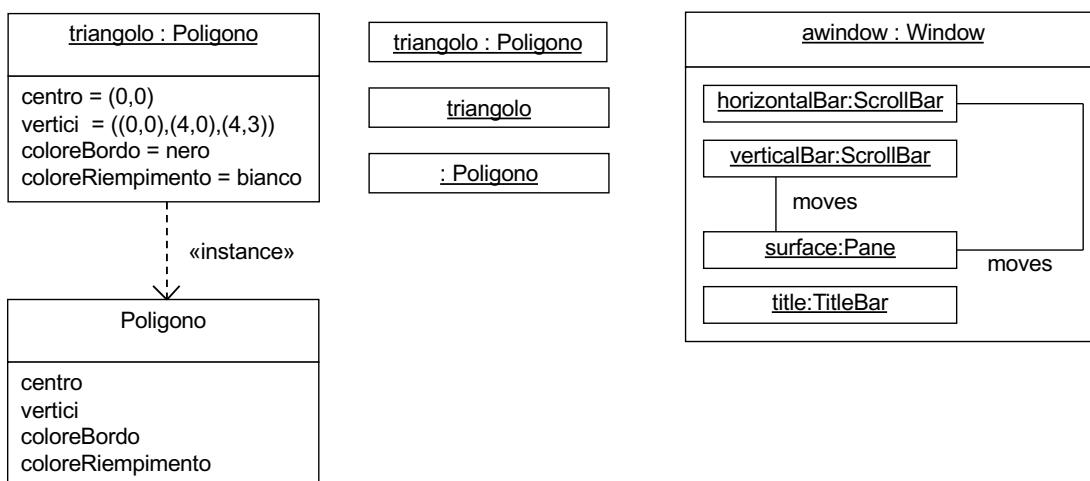


81

# 5

## Diagrammi degli oggetti

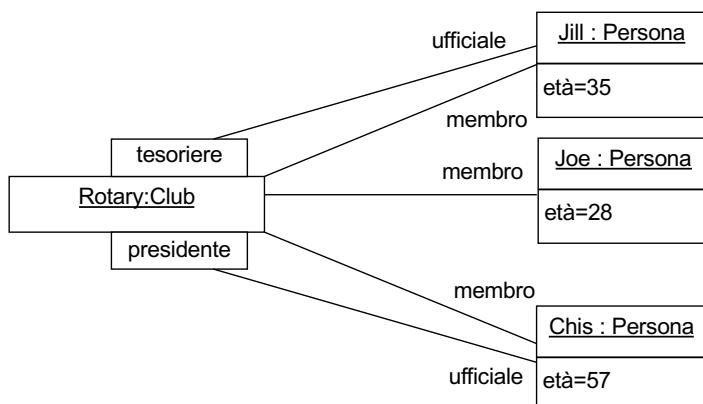
- Un oggetto rappresenta una particolare istanza di una classe.
- Un oggetto *composto* è un oggetto di alto livello che contiene altri oggetti.



82

# Diagrammi degli oggetti

- E' un grafo di istanze di elementi, e rappresenta un'istanza di un diagramma delle classi
  - ⇒ Il suo utilizzo è limitato principalmente a mostrare esempi di strutture dati
  - ⇒ Poiché un diagramma delle classi può contenere anche istanze, un diagramma degli oggetti può essere considerato come un caso particolare di diagramma delle classi in cui compaiono solo oggetti

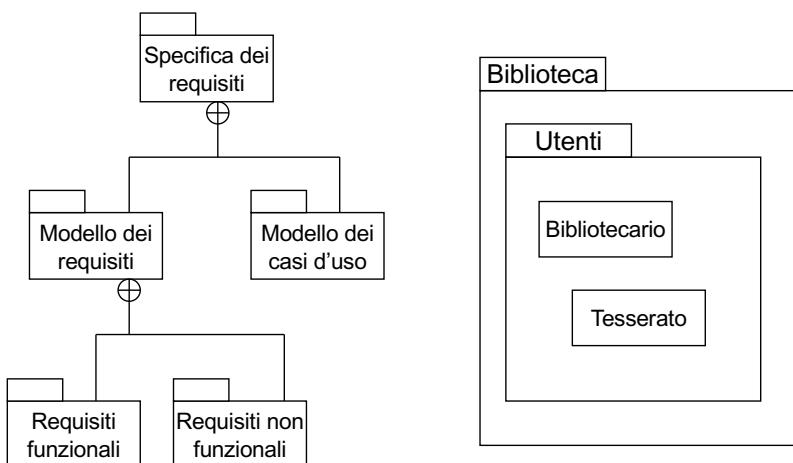


83

# 6

## Diagrammi dei package

- Un package è un raggruppamento di elementi del modello semanticamente correlati
- Relazioni tra package:
  - ⇒ Si possono rappresentare relazioni di **contenimento**; si consiglia di mostrare massimo due livelli. I package annidati vedono lo spazio dei nomi dei package che li contengono, il contrario non è vero



84

# Diagrammi dei package

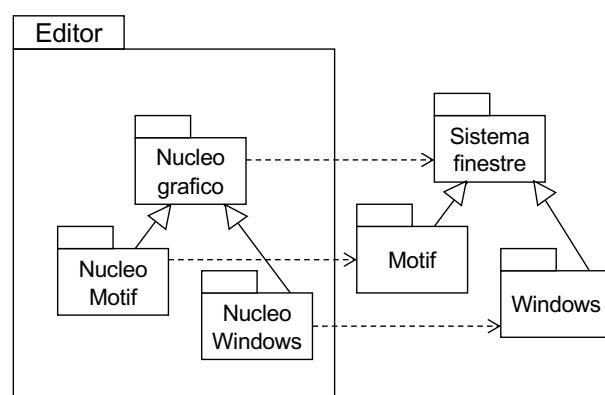
- ➡ Esistono quattro tipi principali di **dipendenza** tra package
  - «use» (default), quando un elemento del package cliente usa in qualche modo un elemento del package fornitore
  - «import», quando gli elementi pubblici dello spazio dei nomi del package fornitore vengono aggiunti come elementi pubblici allo spazio dei nomi del package cliente
  - «access», quando gli elementi privati dello spazio dei nomi del package fornitore vengono aggiunti come elementi privati allo spazio dei nomi del package cliente
  - «trace» rappresenta l'evoluzione di un elemento in un altro elemento più dettagliato



85

# Diagrammi dei package

- ➡ Esiste una **generalizzazione** tra due package quando il package specifico si deve conformare all'interfaccia del package generale



86

## Individuare i package d'analisi

- I package d'analisi sono gruppi di elementi del modello accomunati da forti correlazioni semantiche
- La fonte migliore per individuarli è il **diagramma delle classi**. I migliori candidati per essere raggruppati nello stesso package sono:
  - ⇒ le classi appartenenti a **gerarchie di composizione**
  - ⇒ le classi appartenenti a **gerarchie di specializzazione**
- Anche il **diagramma dei casi d'uso** può servire: uno o più casi d'uso che supportano un processo aziendale o un attore potrebbero indicare un package
- Per minimizzare le interdipendenze si possono poi spostare classi tra package, aggiungere package, eliminare package
- Numero ideali di classi per package: tra 4 e 10
- Conviene evitare la dipendenze circolari

87

## 7

## Diagrammi di interazione

- Rappresentano la struttura dell'interazione tra oggetti durante uno scenario
- Esistono quattro tipi di diagrammi di interazione, ognuno rivolto a un particolare aspetto:
  - ⇒ **Diagramma di sequenza**: enfatizza la sequenza temporale degli scambi di messaggi
  - ⇒ **Diagramma di comunicazione**: enfatizza le relazioni strutturali tra gli oggetti che interagiscono
  - ⇒ **Diagramma di sintesi dell'interazione**: illustra come un comportamento complesso viene realizzato da un insieme di interazioni più semplici
  - ⇒ **Diagramma di temporizzazione**: enfatizza gli aspetti real-time di un'interazione

88

# Terminologia

- Una **interazione** è un'unità di comportamento di un classificatore che ne costituisce il contesto; essa comprende un insieme di messaggi scambiati tra linee di vita all'interno del contesto per ottenere un obiettivo
- Il **contesto** può essere dato dall'intero sistema, da un sottosistema, da un caso d'uso, da un'operazione, da una classe
- Una **linea di vita** rappresenta come un'istanza di un classificatore partecipa all'interazione
- Un **messaggio** rappresenta un tipo specifico di comunicazione istantanea tra due linee di vita in un'interazione, e trasporta informazione nella prospettiva che seguirà una attività

89

# Linee di vita

- Sintassi:



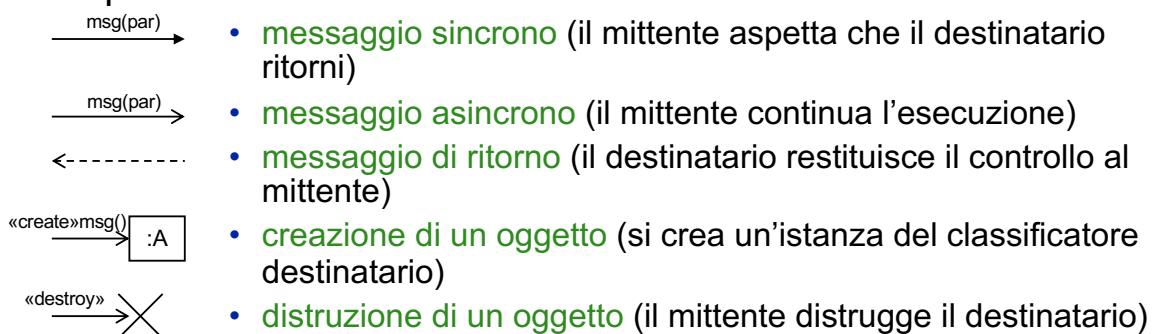
- Sono disegnate con lo stesso simbolo del loro classificatore
- Possono avere una “coda” a forma di riga verticale tratteggiata
- Non rappresentano specifiche istanze del classificatore, ma *modi* in cui le istanze partecipano all'interazione

90

# Messaggi

- ⇒ messaggi di chiamata
- ⇒ messaggi di creazione
- ⇒ messaggi di distruzione
- ⇒ invio di segnali

- Per ogni messaggio di chiamata ricevuto da una linea di vita, deve esistere un'operazione corrispondente nel classificatore di quella linea di vita



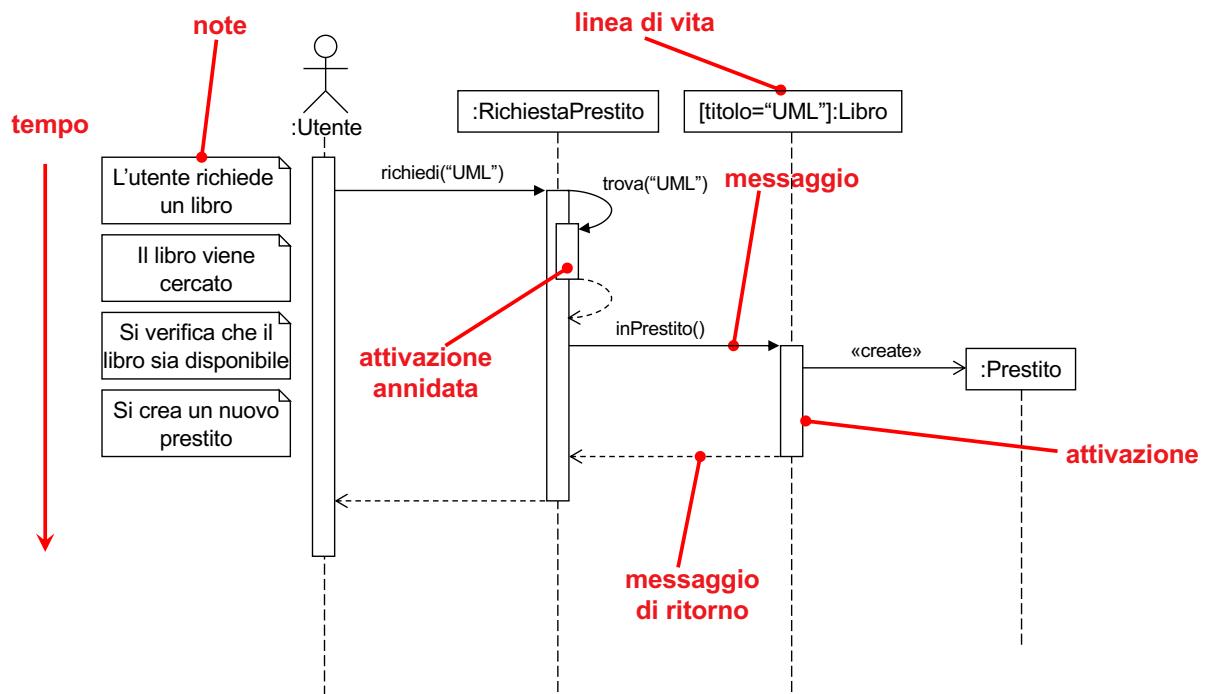
91

# Diagrammi di sequenza

- Mostrano le interazioni tra linee di vita come una sequenza di messaggi ordinati temporalmente
- Sono la forma più ricca e flessibile di diagramma di interazione
  - ⇒ Hanno due **dimensioni**: la dimensione verticale rappresenta il tempo mentre quella orizzontale rappresenta le linee di vita
  - ⇒ Un'**attivazione** mostra il periodo durante il quale una linea di vita esegue un'azione o direttamente o attraverso una procedura subordinata; rappresenta sia la durata dell'azione nel tempo sia la relazione di controllo tra l'attivazione e i suoi chiamanti
  - ⇒ Si possono specificare **nodi decisionali**, **iterazioni**, **attivazioni annidate**
  - ⇒ È consigliato descrivere il flusso tramite un'insieme di **note** poste accanto agli elementi

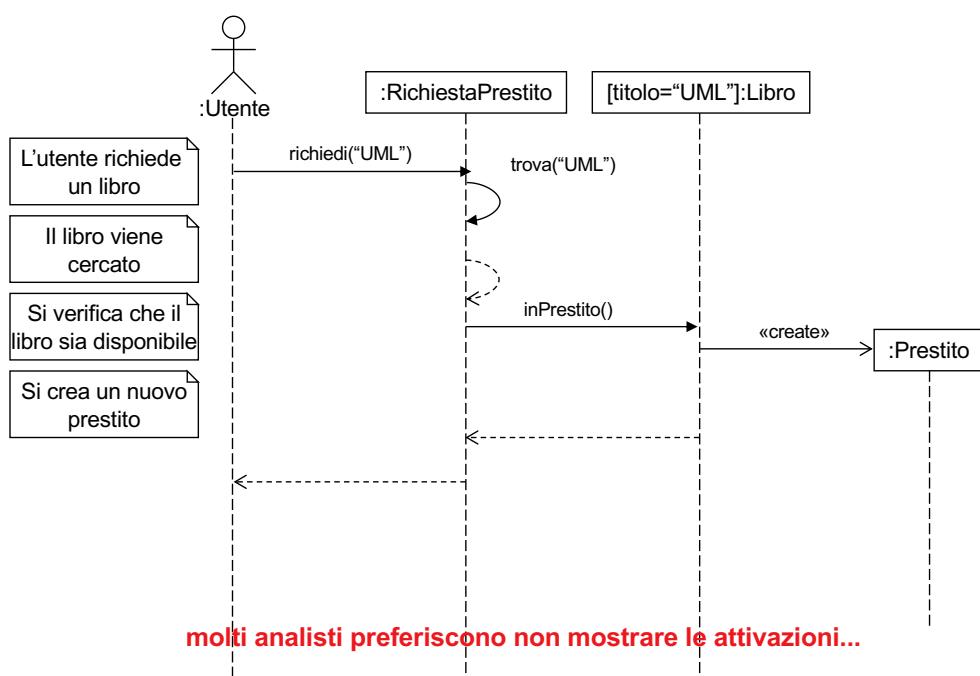
92

# Richiesta prestito



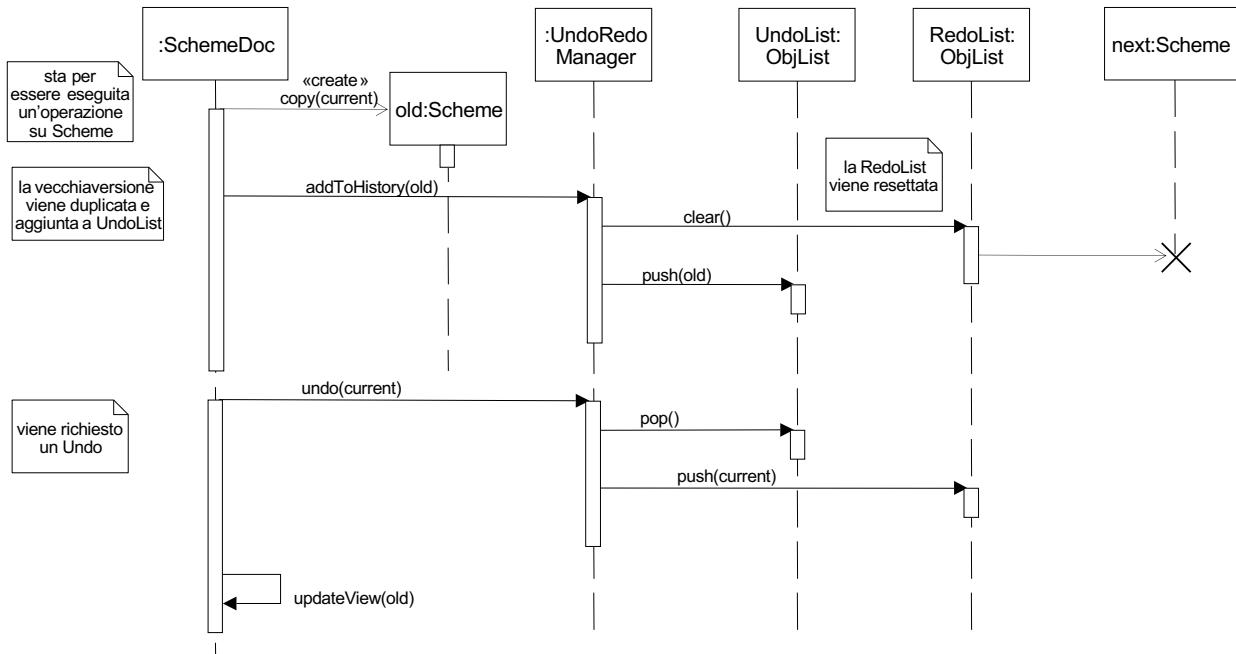
93

# Richiesta prestito



94

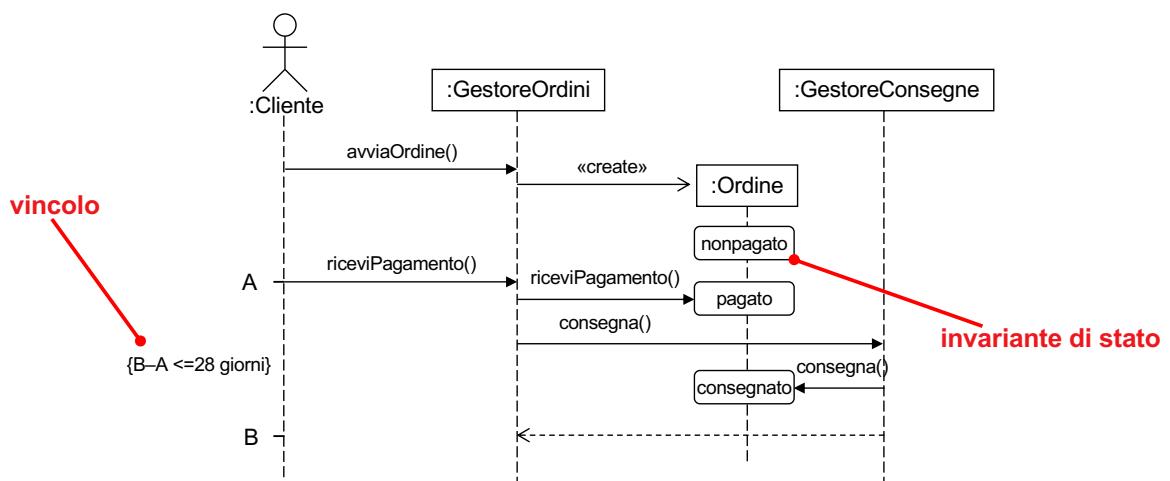
# Gestione undo/redo



95

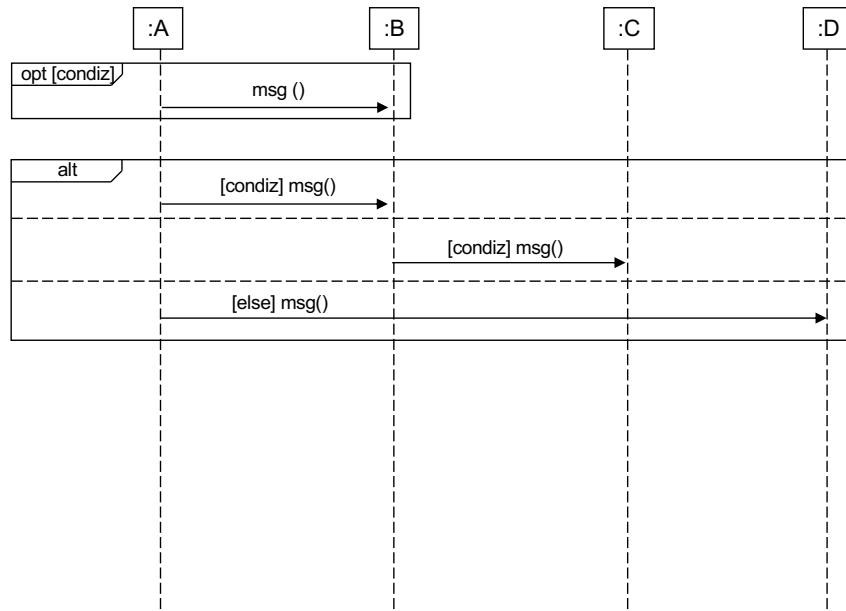
# Invarianti di stato e vincoli

- Quando un'istanza riceve un messaggio, il suo stato può cambiare
- Lo **stato** delle istanze può essere mostrato sulla linea di vita
- Un **vincolo** posto sulla linea di vita indica una condizione sulle istanze che deve essere vera da lì in avanti



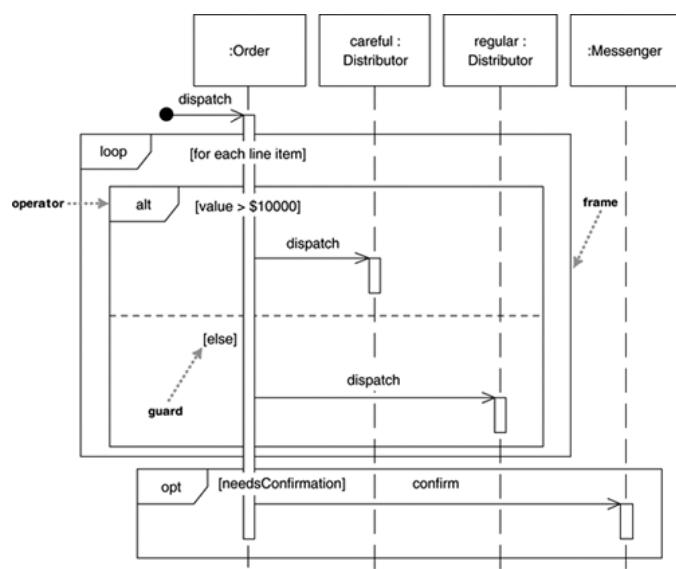
96

# Frammenti combinati



97

# Frammenti combinati

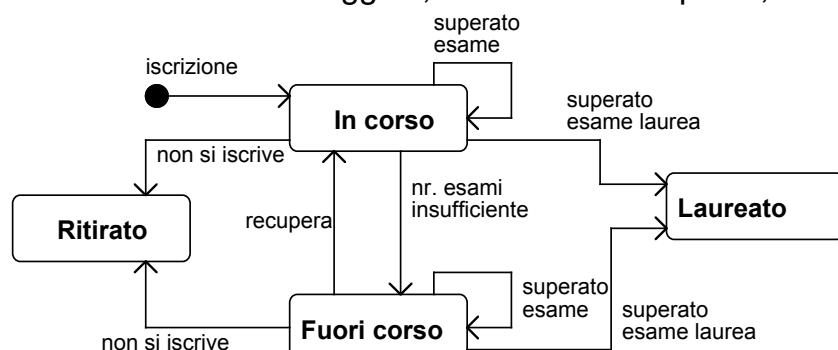


98

# 8

# Diagrammi di stato

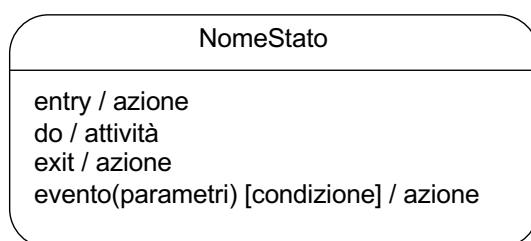
- I diagrammi di stato descrivono in modo esaustivo l'evoluzione temporale delle istanze di un classificatore (classe, caso d'uso, sottosistema) in risposta alle interazioni con altri oggetti
- Ogni classe può avere associato un diagramma di stato
  - ⇒ UML adotta la [notazione di Harel](#), che può esprimere sottostati, stati composti, parallelismo, stati storici, gestione eventi, operazioni, creazione e distruzione di oggetti, marcamenti temporali, ecc.



99

# Stati ed eventi

- Lo **stato** di un oggetto in un certo istante è un'astrazione dell'insieme dei valori dei suoi attributi e dei suoi collegamenti
  - ⇒ Le differenti configurazioni di valori e collegamenti vengono raggruppate in stati a seconda di come incidono sul comportamento macroscopico dell'oggetto
- Un **evento** provoca la transizione tra uno stato e l'altro; un oggetto rimane in uno stato per un tempo finito non istantaneo corrispondente all'intervallo tra due eventi
- Uno stato può contenere azioni e attività:
  - ⇒ Le **azioni** sono operazioni istantanee, atomiche e non interrompibili; sono associate a transizioni attivate da eventi
  - ⇒ Le **attività** sono operazioni che richiedono un certo tempo per essere completate e possono quindi essere interrotte da un evento



100

# Transizioni

- Una **transizione** marca il passaggio di un oggetto da uno stato a un altro, ed è associata a uno o più eventi e, optionalmente, a condizioni e azioni
- Un **evento** avviene a un preciso istante di tempo, e si assume che abbia durata nulla
  - ⇒ Gli eventi possono essere raggruppati in classi, eventualmente descritte da attributi
- Una **condizione** è un'espressione booleana che deve risultare vera affinché la transizione possa avvenire
- Un'**azione** è un'operazione istantanea, atomica e non interrompibile che viene eseguita all'atto della transizione

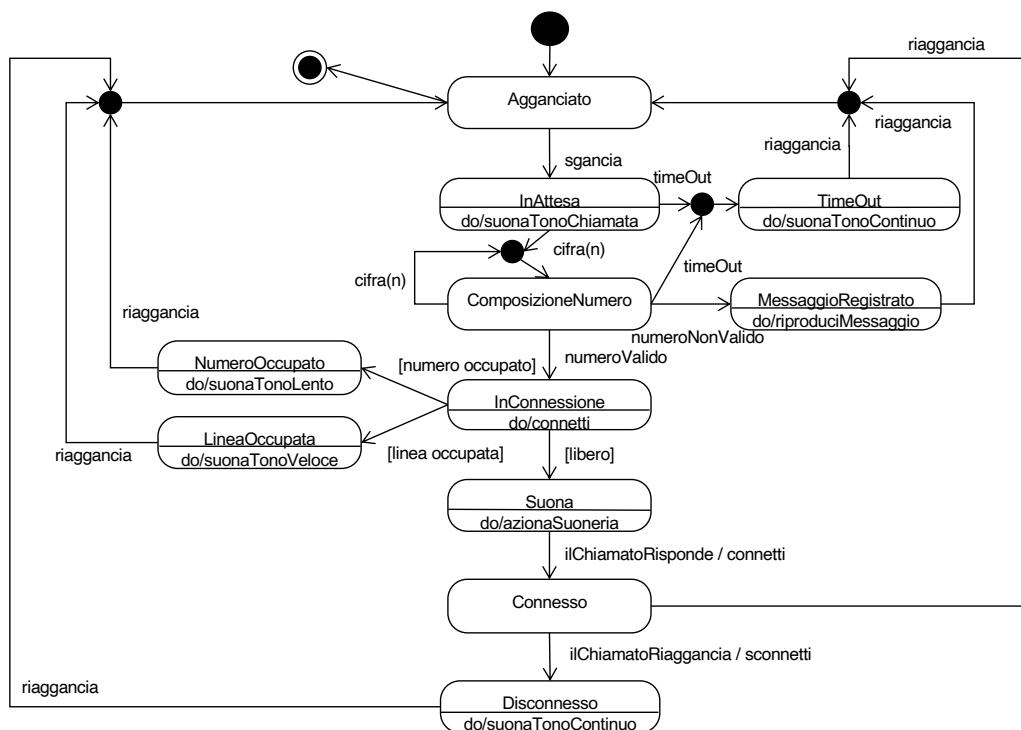
evento(parametri) [condizione] / azione  
→

- Una transizione che esce da uno stato e non riporta alcun evento indica che la transizione avviene al termine dell'attività



101

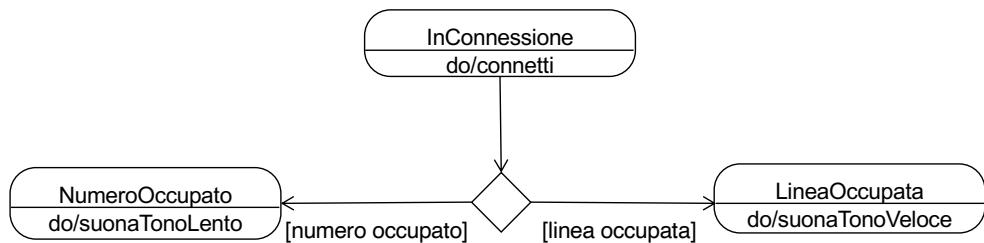
## La linea telefonica



102

# Pseudo-stato di selezione

- Consente di dirigere il flusso nell'automa secondo le condizioni specificate sulle sue transizioni di uscita



103

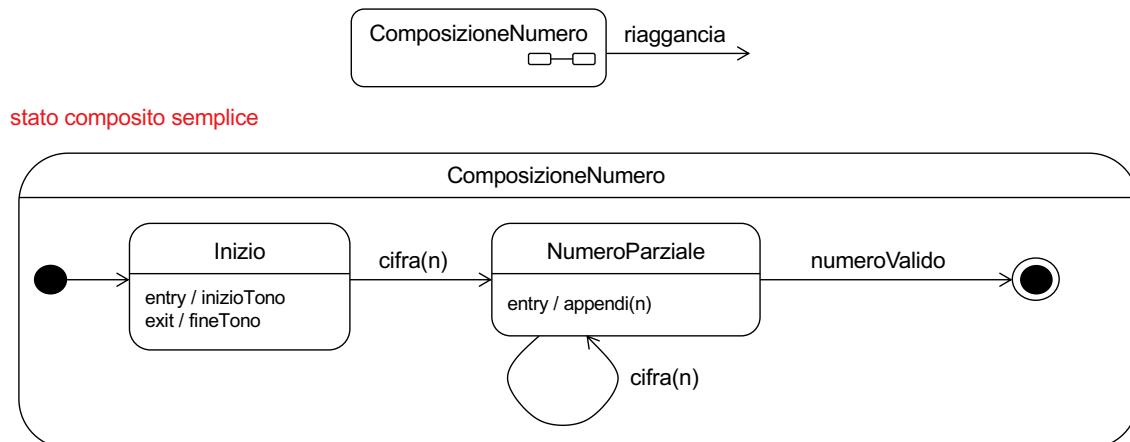
## Tipi di eventi

- **Evento di variazione:** si verifica nel momento in cui una condizione diventa vera
  - ◆ è denotato da un'espressione booleana, ad esempio: bilancio<0
  - ◆ può essere considerato come una condizione verificata continuamente sebbene in realtà verrà controllata solo al variare dei parametri coinvolti
- **Evento di segnale:** si verifica nel momento in cui un oggetto riceve un oggetto *segnaletico* da un altro oggetto
- **Evento di chiamata:** è l'invocazione di una specifica operazione nell'istanza del classificatore che fa da contesto al diagramma
  - ◆ è denotato dalla signature dell'operazione
  - ◆ può essere associato a una sequenza di azioni separate da “;”
- **Evento temporale:** si verifica allo scadere di un periodo di tempo
  - ◆ **when**(data=01/01/2008): specifica il momento della transizione
  - ◆ **after**(10 seconds): specifica che la transizione deve avvenire dopo 10 secondi dall'entrata dell'automa nello stato attuale; è anche possibile specificare il momento in cui inizia a decorrere il periodo aggiungendo una frase del tipo “since...”

104

## Stati composti

- Uno stato composito è uno stato che contiene altri stati annidati, organizzati in uno o più automi
- Ogni stato annidato eredita tutte le transizioni dello stato che lo contiene

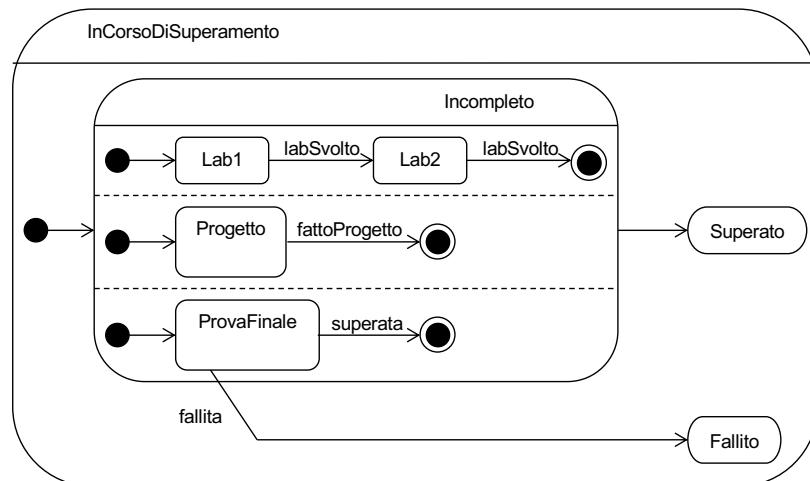


105

## Stati composti

- Lo pseudo-stato finale di un automa viene applicato solo a quell'automa

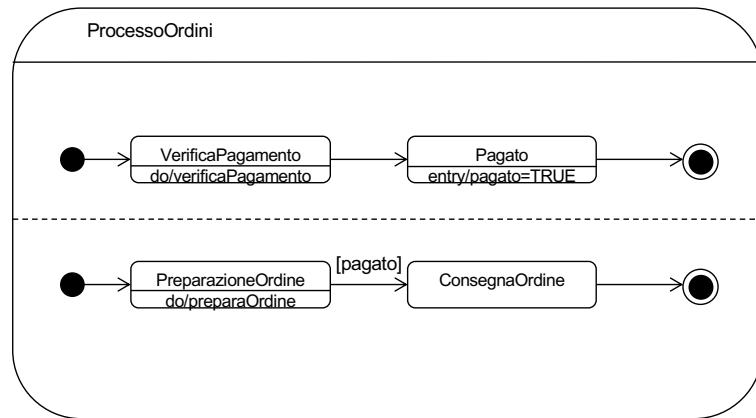
stato composito ortogonale



106

# Comunicazione tra automi

- Si fanno comunicare in modo asincrono i due automi attraverso la variabile “pagato”



107

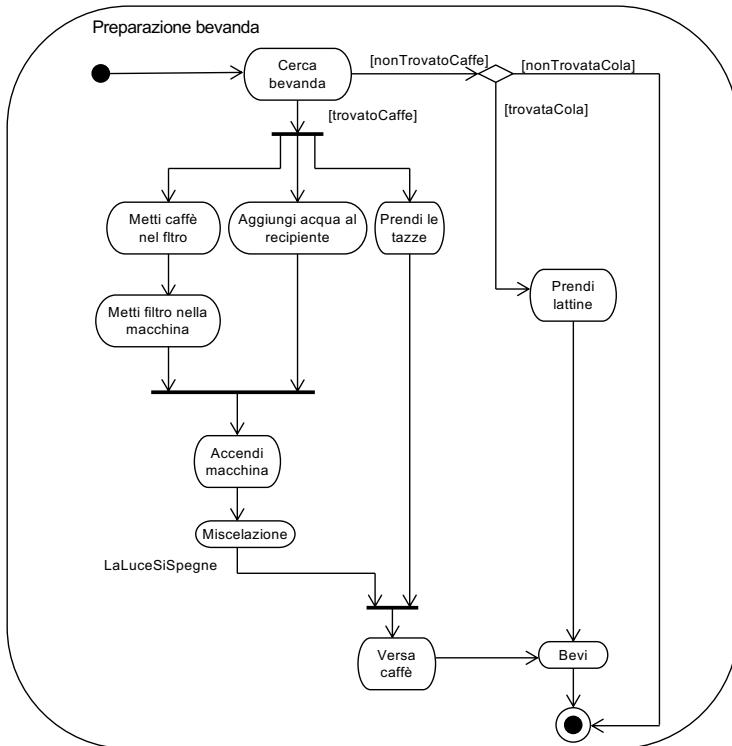
## 9

# Diagrammi di attività

- Modellano un processo come un’attività costituita da un insieme di nodi connessi da archi
- In UML 2 hanno una nuova semantica basata sulle reti di Petri, differenziandosi completamente dai diagrammi degli stati
- Un’attività può essere associata a qualunque elemento di modellazione, che ne diviene il contesto:
  - ⇒ caso d’uso
  - ⇒ operazione
  - ⇒ classe
  - ⇒ interfaccia
  - ⇒ componente
  - ⇒ collaborazione
- I diagrammi di attività possono anche essere usati per modellare efficacemente processi di business e workflow

108

# Preparazione di una bevanda



109

## Attività

- Sono modellate come reti di **nodi** connessi da **archi**
- Categorie di nodi:
  - ⇒ **nodi azione**, che rappresentano compiti atomici all'interno dell'attività
  - ⇒ **nodi controllo**, che controllano il flusso all'interno dell'attività
  - ⇒ **nodi oggetto**, che rappresentano oggetti usati nell'attività
- Categorie di archi:
  - ⇒ **flussi di controllo**, che rappresentano il flusso di controllo attraverso l'attività
  - ⇒ **flussi di oggetti**, che rappresentano il flusso di oggetti attraverso l'attività

110

# Nodi azione

## ☐ Nodo azione di chiamata

⇒ chiama un comportamento

Crea ordine

⇒ chiama un'attività

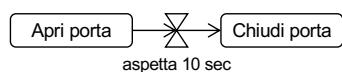
Crea ordine

⇒ chiama un'operazione

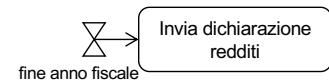
Stampa ordine  
(Ordine::stampa)

## ☐ Nodo azione di accettazione evento temporale

⇒ produce un evento temporale ogni volta che la condizione temporale diventa vera



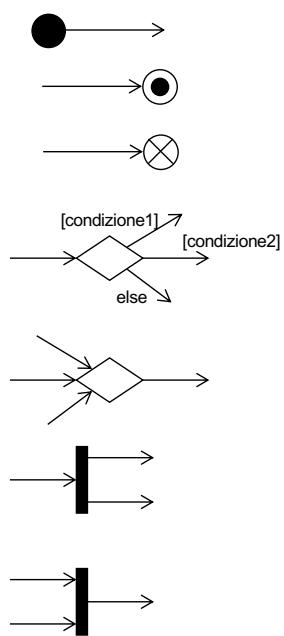
⇒ diventa attivo solo quando si attiva l'arco



111

# Nodi controllo

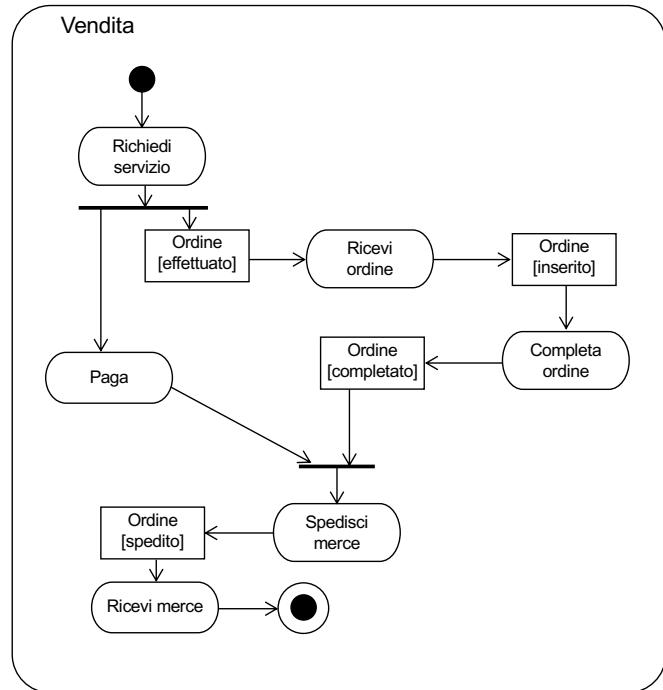
- ☐ **Nodo iniziale:** indica l'inizio del flusso
- ☐ **Nodo finale dell'attività:** termina un'attività
- ☐ **Nodo finale del flusso:** termina uno specifico flusso
- ☐ **Nodo decisione:** divide il flusso in più flussi alternativi
- ☐ **Nodo fusione:** ricongiunge i flussi a valle di un nodo decisione
- ☐ **Nodo biforcazione:** divide il flusso in più flussi concorrenti
- ☐ **Nodo ricongiunzione:** sincronizza flussi concorrenti



112

# Nodi oggetto

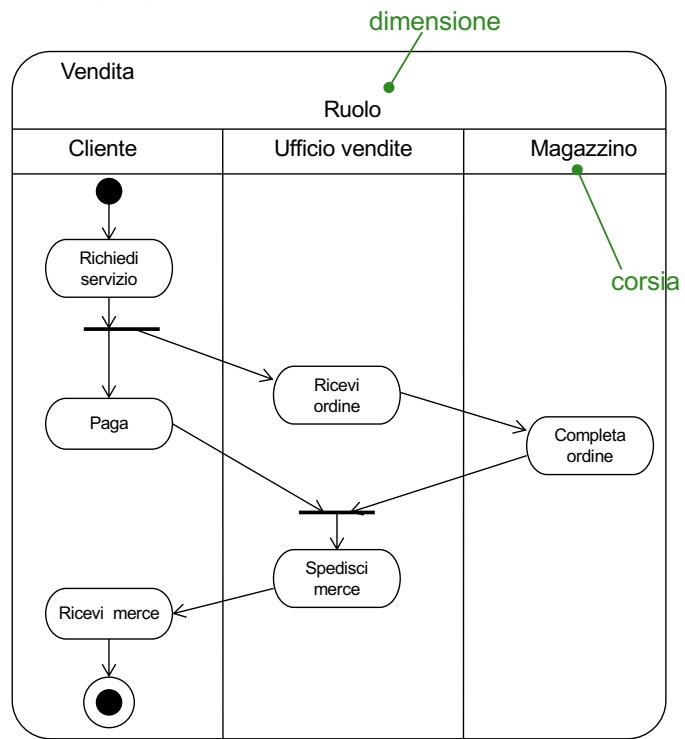
- I nodi oggetto indicano che sono disponibili istanze di una data classe in un punto specifico dell'attività
- Gli archi in entrata e uscita dai nodi oggetto rappresentano flussi di oggetti creati e consumati da nodi azione
- E' possibile rappresentare esplicitamente lo stato di un oggetto



113

# Corsie

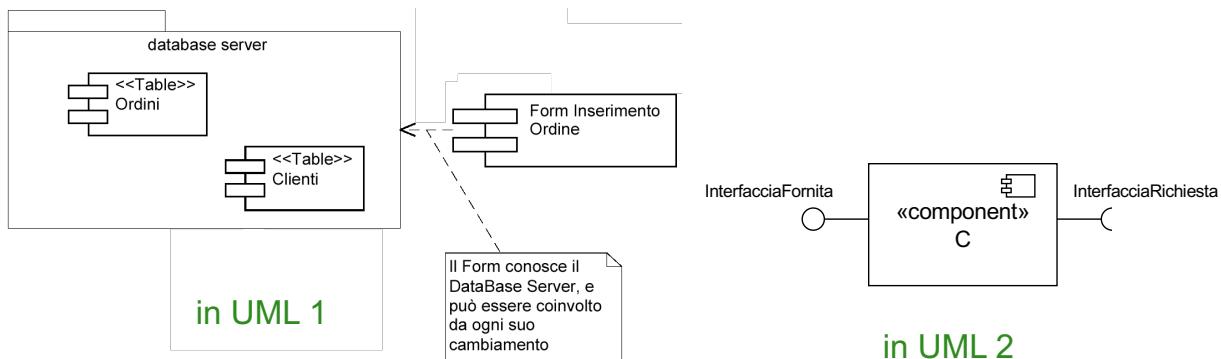
- Le attività possono essere partizionate in **corsie** che raggruppano insiemi di azioni correlate
- Le corsie possono corrispondere a
  - ⇒ casi d'uso
  - ⇒ classi
  - ⇒ componenti
  - ⇒ unità organizzative
  - ⇒ ruoli
- La semantica di ogni insieme di corsie è descritta da una **dimensione**



114

# 10 Diagramma dei componenti

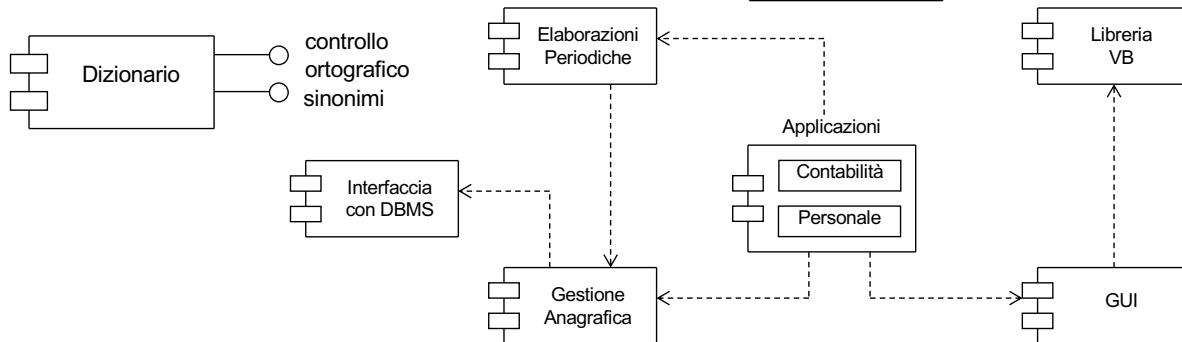
- Mostra i componenti e le loro interdipendenze
- Un **componente** è una parte modulare di un sistema che incapsula i suoi contenuti (black box)
- I componenti possono avere attributi e operazioni, e possono partecipare ad associazioni e generalizzazioni



115

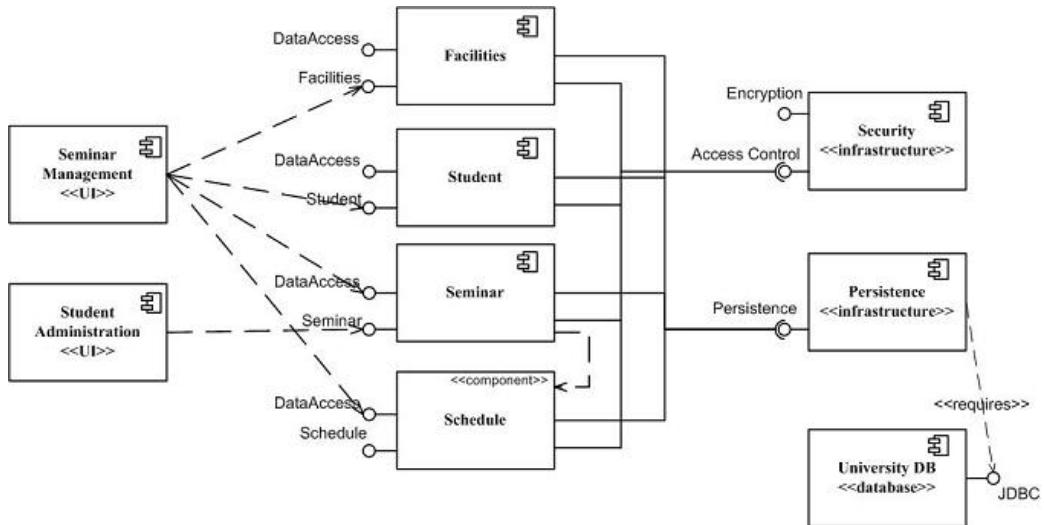
## Componenti

- I componenti possono contenere oggetti, ad indicare che essi ne sono parte integrante
- I componenti sono connessi tra loro mediante **dipendenze**, possibilmente tramite **interfacce**, ad indicare che un componente usa i servizi di un altro componente



116

# Esempio

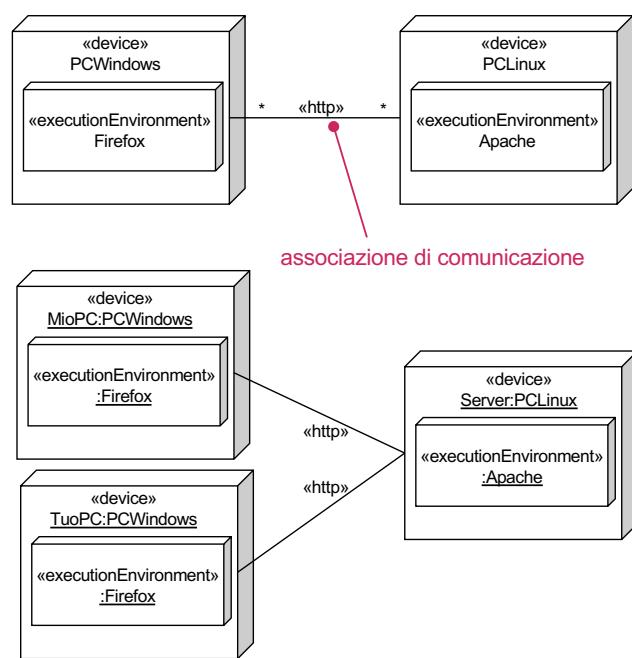


117

## 11

## Diagramma di deployment

- Specifica l'hardware su cui verrà eseguito il software e il modo in cui il software è dislocato sull'hardware
- Può avere due forme:
  - ⇒ **descrittore**, che contiene nodi, relazioni tra nodi e manufatti; modella *tipi* di architetture
  - ⇒ **istanza**, che contiene istanze di nodi, di relazioni tra nodi e di manufatti; modella un deployment dell'architettura su un particolare sito



118

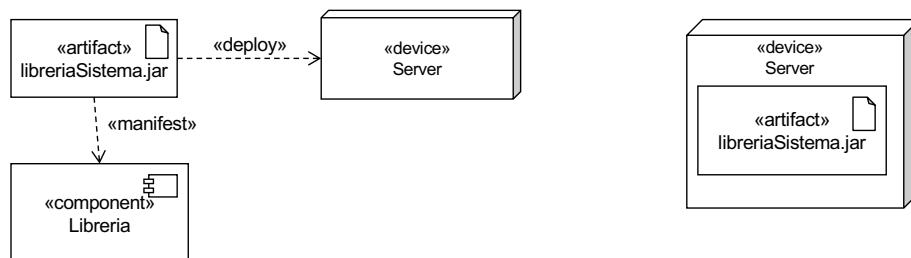
# Nodi

- Un *nodo* rappresenta un tipo di risorsa computazionale su cui i manufatti possono essere dislocati per l'esecuzione
- Due stereotipi standard:
  - ⇒ «device» rappresenta un tipo di periferica fisica, per esempio un PC
  - ⇒ «executionEnvironment» rappresenta un tipo di ambiente software di esecuzione, per esempio un web server
- I nodi possono essere annidati in nodi
- Un'associazione tra nodi rappresenta un canale di comunicazione tra di essi
- Si possono usare ulteriori stereotipi e icone per aumentare la leggibilità del diagramma

119

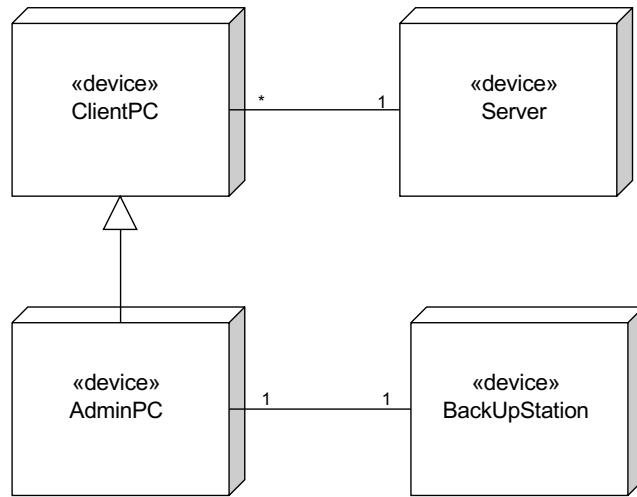
# Manufatti

- Un manufatto rappresenta un'entità concreta del mondo reale, per esempio:
  - ⇒ file sorgenti
  - ⇒ file eseguibili
  - ⇒ script
  - ⇒ tabelle di database
  - ⇒ documenti
  - ⇒ modelli UML
- I manufatti vengono dislocati sui nodi



120

# Esempio



121

12

## Benefici di UML

- Superamento della "guerra dei metodi"
  - ⇒ decine di metodi di analisi e disegno proposti e praticati
  - ⇒ difficoltà per chi vuole passare all'approccio object-oriented: qual è il metodo migliore?
  - ⇒ quale strumento scegliere, se non c'è chiarezza nel campo dei metodi?
- Risposta ai problemi legati allo sviluppo di sistemi complessi con ambienti visuali
  - ⇒ ritorno di attenzione sul processo di lavoro e sugli approcci utilizzati, non solo sulle tecnologie
  - ⇒ il metamodello comune favorisce le possibilità di comunicazione tra strumenti di supporto alla progettazione, e più in generale tra i diversi ambienti utilizzabili dai progettisti nello sviluppo

122

# Complessità

- Il metamodello di UML è molto complesso, perché ha l'ambizione di poter rappresentare qualunque tipo di sistema software, a livelli di astrazione differenziati
- Il numero dei diagrammi è elevato, e per molti diagrammi è possibile scegliere tra forme di rappresentazione leggermente diverse tra loro
- UML non suggerisce, né tantomeno prescrive una sequenza di utilizzo dei diversi diagrammi, lascia anzi molte strade aperte, tra le quali i progettisti sono liberi di scegliere

123

# Personalizzazioni

- Le realtà che sviluppano software sono molto eterogenee: chi sviluppa per conto proprio non ha le stesse esigenze di documentazione e comunicazione di chi opera in un gruppo di lavoro all'interno di un'azienda
- Tra aziende diverse possono esserci differenze anche notevoli nel livello di formalizzazione richiesto ai progettisti, nelle tecniche da adottare, negli approcci da seguire, nel tipo di documentazione da produrre
- I progetti non sono tutti uguali: variano per dimensioni, per tipologia, per criticità, e per molti altri fattori
- UML può essere utilizzato da tutti, perché è sufficientemente complesso per potersi adattare a tutte le esigenze....
- ...ma non ha senso che tutti utilizzino UML esattamente nello stesso modo: per scegliere il percorso "ottimale", e quali diagrammi utilizzare davvero tra tutti quelli che UML mette a disposizione, è necessario verificare quali siano le esigenze da soddisfare

124

## Quindi:

- UML è uno **standard**, e questo è un bene (uniformità nei concetti e nelle notazioni utilizzate, interoperabilità tra strumenti di sviluppo, indipendenza dai produttori, dalle tecnologie, dai metodi)
- UML è **articolato**: può rappresentare qualunque sistema software, a diversi livelli di astrazione
- UML è **complesso**: va adattato (“ritagliato”) in base alle specifiche esigenze dei progettisti e dei progetti, utilizzando solo ciò che serve nello specifico contesto