

# 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

CASE è l'acronimo di "Computer-Aided Software Engineering". Si riferisce a una categoria di strumenti software progettati per supportare e automatizzare le diverse fasi del ciclo di vita dello sviluppo del software, tra cui analisi, progettazione, codifica, test e manutenzione.

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)

Un metamodello è un modello che definisce le regole e le strutture con cui sono creati altri modelli. In altre parole, un metamodello descrive la struttura di base dei modelli stessi.

3

## Generalità

- UML fornisce i costrutti per le seguenti fasi dello sviluppo dei sistemi software:
  - ⇒ Analisi dei requisiti **tramite i casi d’uso** per raccogliere e rappresentare i requisiti del sistema (Diagramma dei Casi d’Uso).
  - ⇒ Analisi e progetto OO Diagramma delle Classi e Oggetti
  - ⇒ Modellazione dei componenti Diagramma dei Componenti
  - ⇒ Modellazione della struttura e della configurazione Diagramma di Deployment
- 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 Ad esempio: Una classe potrebbe apparire sia in un diagramma delle classi che in un diagramma di sequenza.
- 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

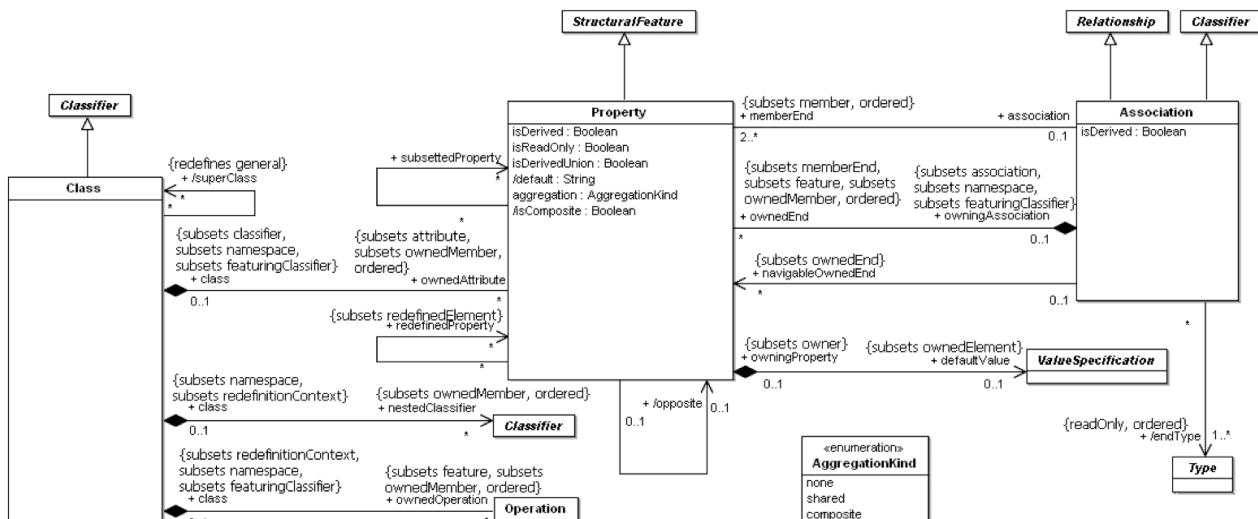
Un certo elemento può comparire in più diagrammi ma è univoca la sua definizione all'interno del modello

- Modello: Una rappresentazione completa, astratta e unificata di un sistema software, che contiene tutti gli elementi e le loro definizioni.  
- Diagramma: Una rappresentazione grafica di un sottoinsieme del modello, che offre una visione parziale e focalizzata su uno specifico aspetto del sistema.

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** Sono la descrizione testuale della semantica di un elemento
- **ornamenti** Rendono visibili gli aspetti particolari della specifica dell'elemento
- **distinzioni comuni** Sono regole o convenzioni che UML usa per differenziare concetti simili.
- **meccanismi di estendibilità** Permettono di adattare UML alle necessità specifiche di un progetto o di una metodologia.

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

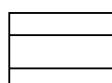
7

## Entità

Sono gli elementi di modellazione

### Strutture:

- Classe



- Interfaccia



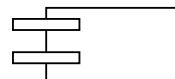
- Collaborazione



- Caso d'uso



- Componente



- Nodo



### Comportamenti:

- Interazione

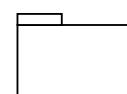


- Stato



### Raggruppamenti:

- Package



### Informazioni:

- Annotazione



8

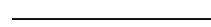
# Relazioni

*Legano tra loro le entità*

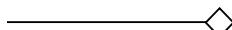
- Dipendenza



- Associazione



- Aggregazione



- Contenimento (solo per Package Diagram)



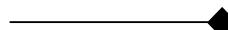
- Generalizzazione



- 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:**

- mostra un insieme di oggetti di interesse e le loro relazioni
- 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

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:

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

mostra i package e le loro relazioni di dipendenza, contenimento e specializzazione

a dei casi d'uso  
a degli stati

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:

- Diagramma dei casi d'uso
- Diagramma dell'architettura software del sistema
- Diagramma di interazione
  - ✓ Diagramma di sequenza
  - ✓ Diagramma di comunicazione
  - ✓ Diagramma di sintesi dell'interazione
  - ✓ Diagramma dei tempi

descrive l'architettura software del sistema

a degli stati  
a di attività

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

*T'architettura*  
descrive 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**

rappresenta la struttura interna di un classificatore complesso, come una classe o un componente. Questo diagramma illustra come le parti interne del classificatore collaborano tra loro e con l'esterno attraverso porte e connettori.

## 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

per modellare e comprendere le interazioni tra utenti e sistema, garantendo che le funzionalità sviluppate rispondano alle reali esigenze degli utilizzatori.

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: **descrive le sequenze eventi-azioni-transizioni di una funzione**
- Diagramma dei componenti
- Diagramma dei package
- 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

19

# Diagrammi

Sono viste sul modello UML

## Statici:

- Diagramma delle classi: **mostra le interazioni tra gli oggetti durante scenari di funzionamento del sistema**
- 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

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

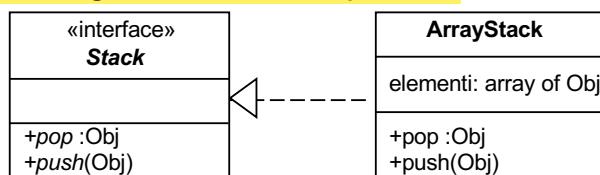
### □ Classificatore/istanza

- ⇒ 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

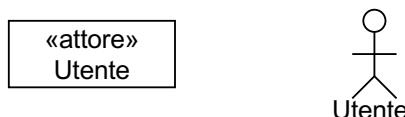
- ⇒ Separa "cosa" un oggetto fa da "come" lo fa
- ⇒ Un'interfaccia definisce un contratto che ciascuna sua implementazione garantisce di rispettare



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



- ⇒ Consente di aggiungere informazioni supplementari agli elementi del modello.
- ### 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

Questo modello suddivide l'architettura in cinque viste distinte, ciascuna focalizzata su specifici aspetti del sistema e rivolta a differenti utenti.

# 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

Si concentra sulla struttura statica del sistema

⇒ Stabilisce la terminologia del dominio del problema sotto forma di classi e oggetti, illustrando come essi implementano il comportamento richiesto

## □ Vista dei processi

Si concentra sugli aspetti dinamici del sistema: descrive come i componenti interagiscono durante l'esecuzione.

⇒ 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

Si concentra su come le entità definite nella vista logica vengono effettivamente eseguite nel tempo, includendo thread, processi e meccanismi di comunicazione.

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

Mostrano gli attori, ovvero chi o cosa interagisce con il sistema.

## Diagrammi dei casi d'uso

Il diagramma dei casi d'uso non è esaustivo.



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 Si concentrano su cosa il sistema fa per gli utenti, ignorando dettagli interni su come queste funzionalità vengono implementate.
  - 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

1) Attori: Sono entità esterne al sistema, che interagiscono con esso per ottenere un risultato.

2) Casi d'Uso: Ogni caso d'uso rappresenta un'azione o una funzionalità che il sistema offre agli attori.

3) Relazioni:

- Associazione: una linea collega un attore a uno o più casi d'uso per rappresentare l'interazione.

- Inclusione: un caso d'uso può "includere" un altro.

- Estensione: un caso d'uso opzionale che estende uno principale

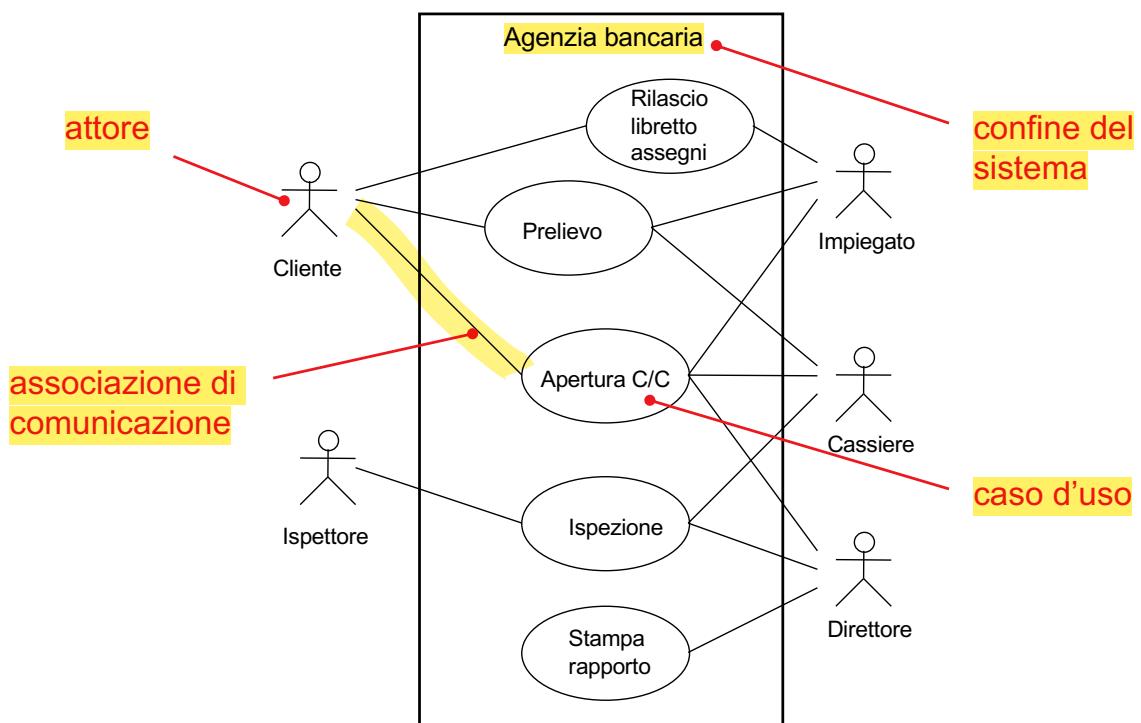
4) Sistema: È rappresentato come un rettangolo che contiene tutti i casi d'uso, con gli attori posizionati all'esterno.

## 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

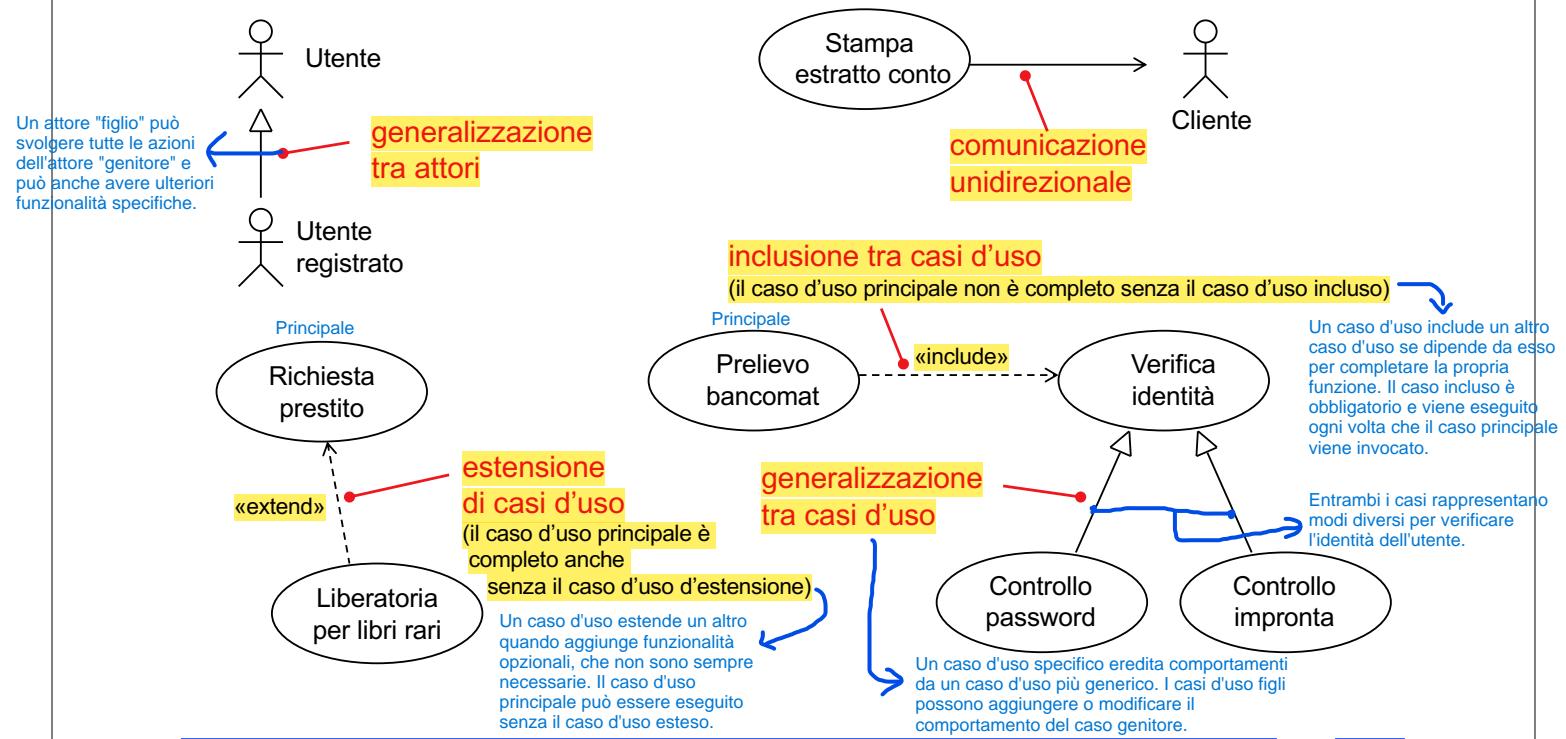
Un caso d'uso descrive una serie di passi che avvengono durante un'interazione tra un attore e il sistema per raggiungere un obiettivo specifico.

# 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

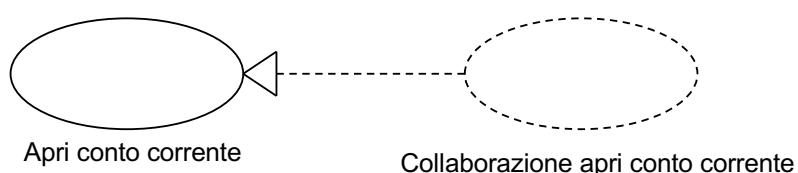
## Specifiche del caso d'uso

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

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 è una descrizione astratta di un gruppo di oggetti che condividono:  
Proprietà comuni (rappresentate dagli attributi).  
Comportamenti comuni (rappresentati dalle operazioni o metodi). Relazioni comuni (con altre classi).

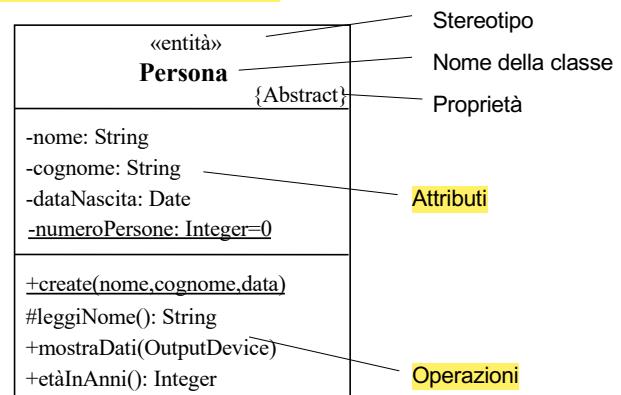
↳ Una **classe** descrive un gruppo di oggetti con proprietà, comportamento e relazioni comuni

↳ È una caratteristica che descrive parte dello stato di un oggetto di una classe.

↳ 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

Quando un'operazione viene eseguita, essa agisce su un oggetto specifico della classe.



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

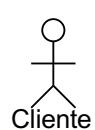
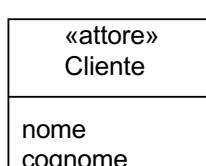
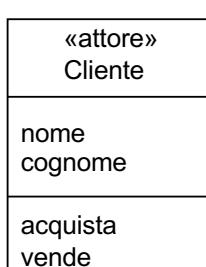
- in
- out
- inout
- return (si usa quando l'operazione restituisce più valori)

### Ambito

- istanza
- classe

43

# Diversi livelli di astrazione



44

# Le relazioni tra classi

(Ereditarietà)

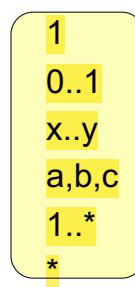
- **Generalizzazione** →
- **Associazione** ——————
- **Dipendenza** ----->
- **Aggregazione** —————— ◊
- **Composizione** —————— ♦
- **Raffinamento** ----->

45

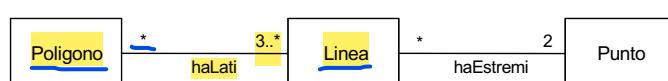
## Associazione

- 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



Una Casa è in 0 o 1 Città  
Una Città ha 1 o più Case

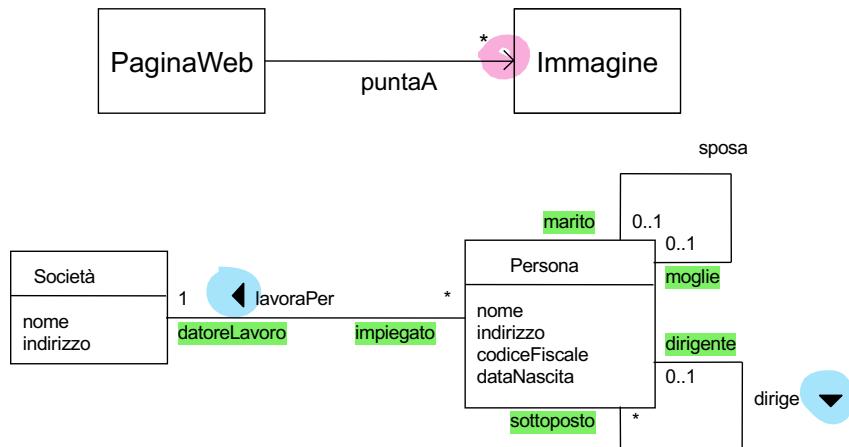


Un Poligono ha 3 o più Lati  
Un Lato è lato di 0 o più Poligoni

46

# Associazione

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



47

- Cosa succede SENZA classe associativa?  
Può esistere più di una relazione tra la stessa persona e azienda, ma non si ha modo di distinguere queste relazioni o arricchirle con dettagli.  
- Cosa succede CON la classe associativa?  
Non è possibile avere due istanze della classe associativa Posizione per la stessa persona e azienda. Se si tenta di creare una relazione duplicata, il modello diventa invalido.

# Associazione

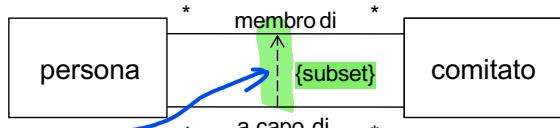
- E' possibile specificare **vincoli** e **classi associative**

non è possibile creare due relazioni duplicate per la stessa persona e la stessa azienda.

*I'identità delle istanze della classe associativa è stabilita solo dalle identità degli oggetti alle sue estremità*

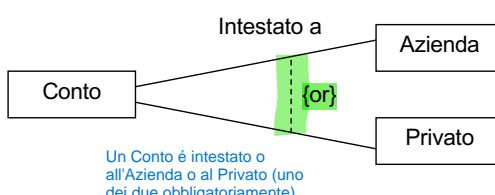
Quando usi una classe associativa, l'istanza della relazione è univoca, poiché:

- L'identità della classe associativa è determinata dagli oggetti alle estremità della relazione.
- Non possono esistere due istanze della classe associativa con gli stessi oggetti collegati.

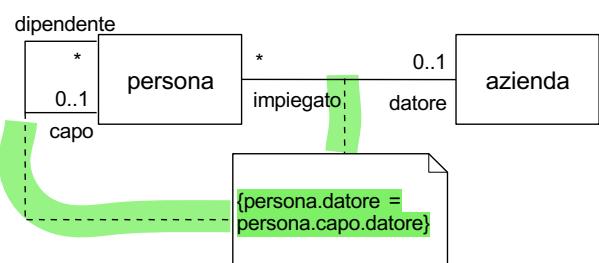


Il vincolo {subset} indica che il ruolo di capo è un sottoinsieme del ruolo di membro.

Una persona può essere capo di un comitato solo se è già membro di quel comitato. In altre parole, non può esistere una persona come capo di un comitato senza essere anche un membro dello stesso.



Un Conto è intestato o all'Azienda o al Privato (uno dei due obbligatoriamente)

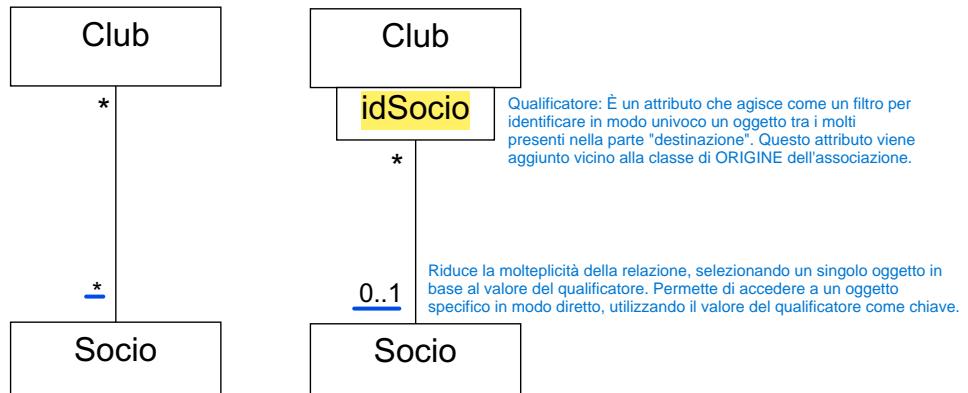


Questo vincolo assicura che il datore di lavoro di un impiegato sia lo stesso datore per cui lavora il suo capo. In altre parole, una persona può essere capo di un'altra solo se entrambe lavorano per la stessa azienda.

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

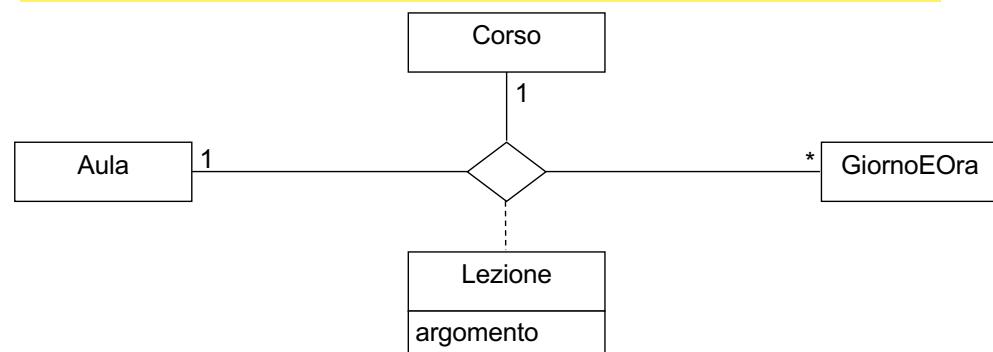


49

# 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 (Esempio: Un Aula e Un GiornoEOra hanno un Corso)
  - I numeri di istanze dell'associazione quando è fissato un solo oggetto sono implicitamente assunti essere tutti a "molti" (Esempio: Un Corso ha 0..\* o 1..\* Lezioni)

Quando hai un'associazione n-aria che collega più classi, ad esempio Aula, Corso, e GiornoEOra, ogni ruolo (cioè ogni classe) ha una molteplicità associata che rappresenta il numero di istanze di quella classe che possono essere collegate a una singola istanza dell'associazione.

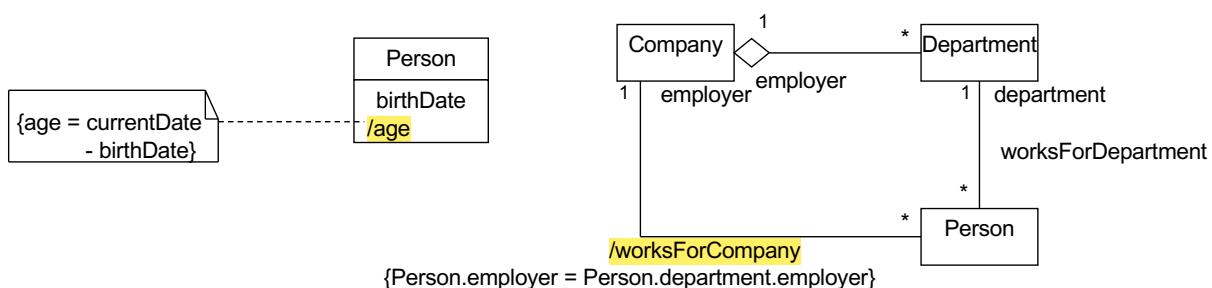


50

## Elementi derivati

(Può essere sia un Attributo sia un'Associazione)

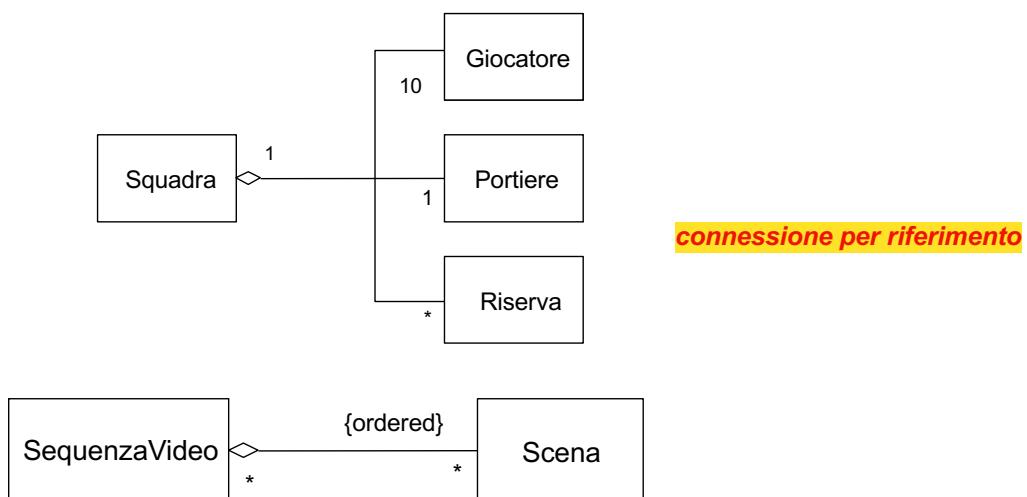
- Un **elemento derivato** può essere calcolato a partire da un altro ma viene mostrato, per motivi di chiarezza o per scelte di progettazione, nonostante non aggiunga alcuna ulteriore informazione semantica
  - ⇒ viene indicato posizionando uno slash prima del suo nome
  - ⇒ i dettagli su come calcolarlo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

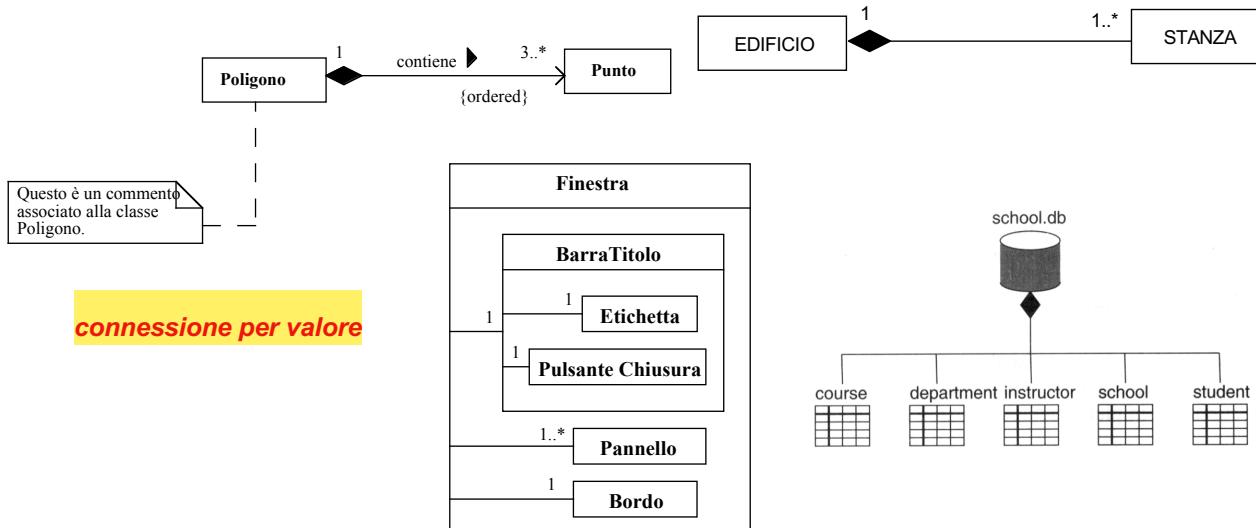
# Composizione

- E' un'aggregazione in cui il tutto "possiede" le sue parti

(Se viene cancellato il "tutto", si cancellano automaticamente anche le parti)

⇒ Le parti esistono solo in relazione al tutto

⇒ Ogni parte appartiene a esattamente un tutto



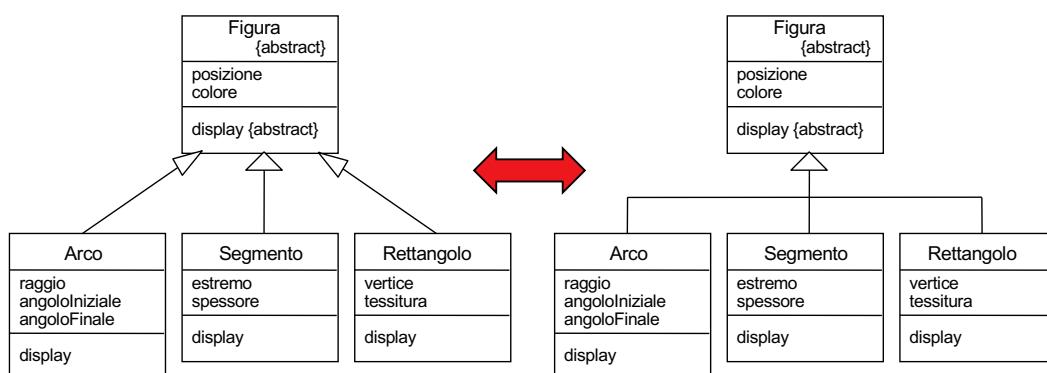
53

# Generalizzazione

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

associazioni

se la superclasse ha una relazione con un'altra classe, le sottoclassi ereditano anche quella relazione, senza doverla ridefinire



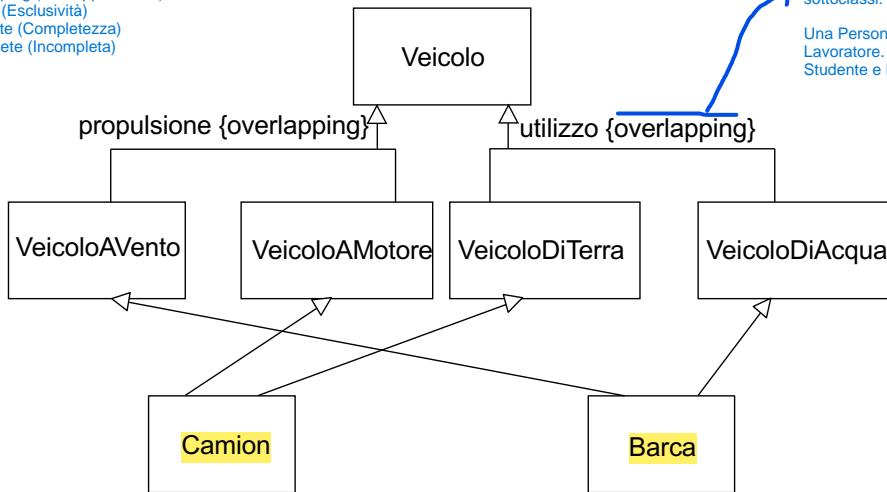
54

- Overlapping: Le istanze della superclasse possono appartenere a più sottoclassi contemporaneamente.
- Disjoint: Le istanze della superclasse appartengono a una sola delle sottoclassi, non possono sovrapporsi.
- Complete: Ogni istanza della superclasse deve appartenere a una delle sottoclassi definite.
- Incomplete: Alcune istanze della superclasse possono non appartenere a nessuna sottoclasse.

## Generalizzazione

- E' supportata l'**ereditarietà multipla** Una classe può estendere più di una classe base (superclasse).
- Possono essere indicati **insiemi di generalizzazione e vincoli** (*overlapping, disjoint, complete, incomplete*)

- 1a) Overlapping (Sovrapposizione)
- 1b) Disjoint (Esclusività)
- 2a) Complete (Completezza)
- 2b) Incomplete (Incompleteta)

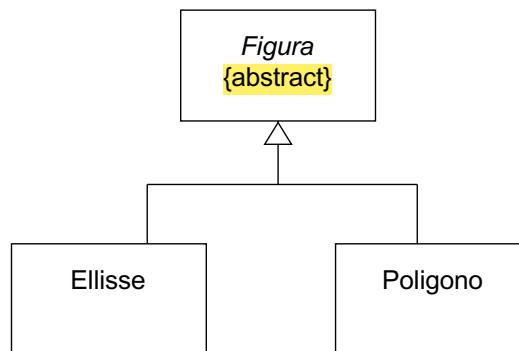


Esempio pratico:  
Immagina di avere una superclasse Persona e due sottoclassi: Studente e Lavoratore.  
Una Persona può essere sia uno Studente che un Lavoratore. Quindi, una Persona può appartenere a Studente e Lavoratore contemporaneamente.

55

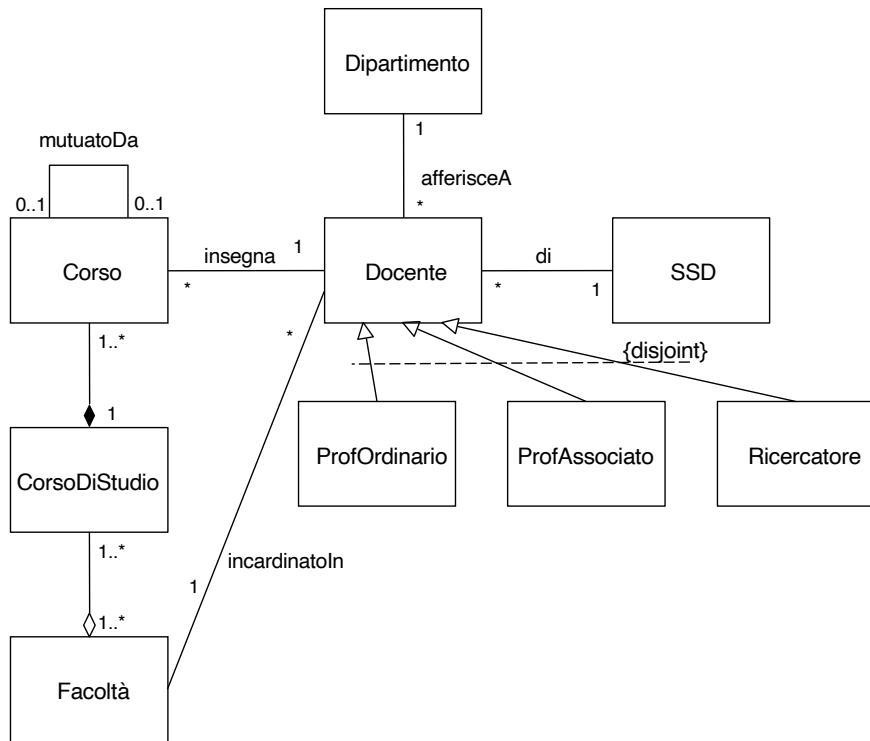
## Classi astratte

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



56

# Un esempio

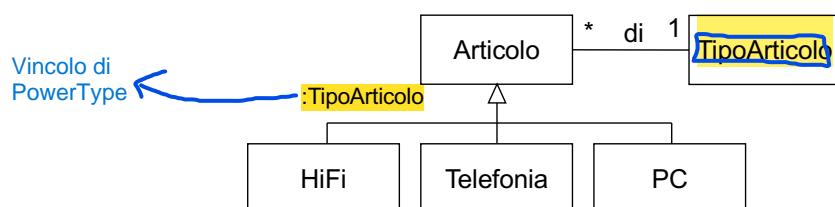


57

# Powertyping

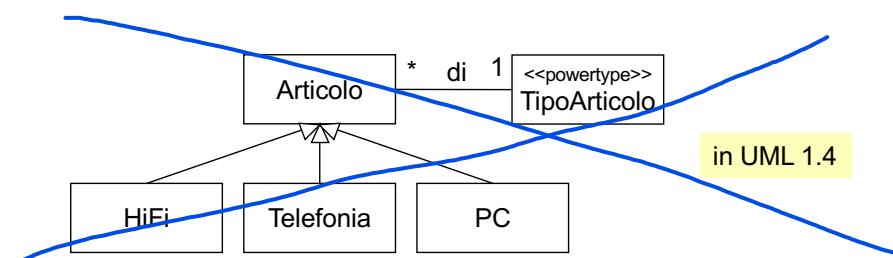
VIENE USATO DI RADO

- Un **powertype** è una (meta)classe le cui istanze sono classi che specializzano un'altra classe, cioè le istanze della powertype sono i nomi delle classi specializzate della gerarchia a cui è legata



in UML 2

Un attributo nelle classi specializzate deve avere valore per ogni istanza della classe, mentre nel PowerType l'attributo viene definito per ogni Tipo di Classe.



in UML 1.4

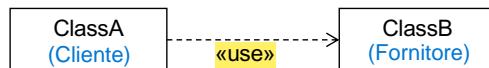
58

Si usano quando si vuole ragionare sulla manutenibilità (noi usiamo per analisi, quindi non serve a noi)

# Dipendenza

In Diagramma Classi, non si usa Dipendenza nei nostri esercizi

- 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 fornitore**, ma non ha una struttura interna che dipende da quest'ultima
  - ⇒ Lo stereotipo più comunemente usato è «use» (Classe A usa Classe B: usa i suoi servizi)



Una classe cliente può dipendere da un'altra classe fornitore, ma senza che la sua struttura interna dipenda direttamente dalla classe fornitore.

- ⇒ Più specificamente, si può rappresentare il fatto che un'operazione della classe cliente ha argomenti che appartengono al tipo di un'altra classe (Operazione in A usa come parametri istanze di B)

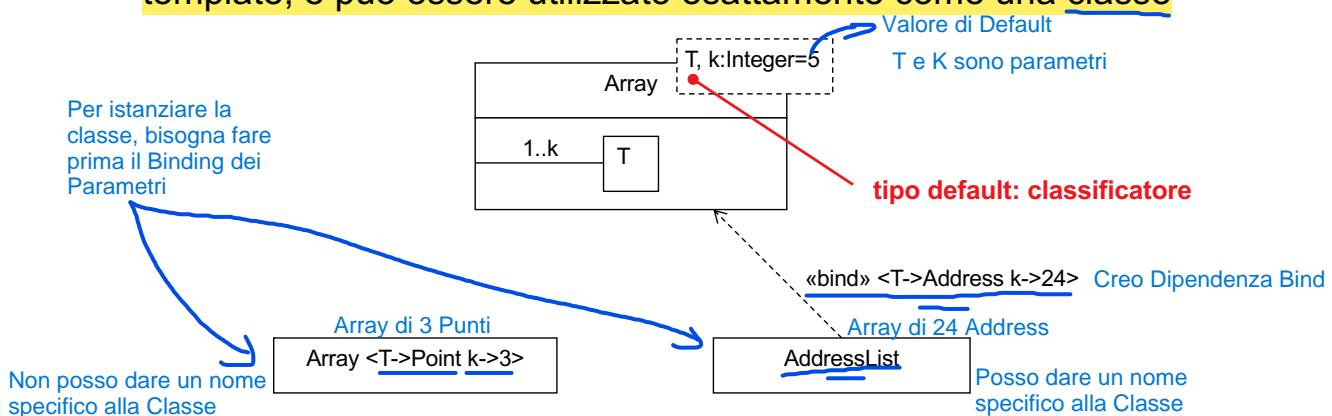


59

# Template

NOI NEI COMPITI NON LI USIAMO

- 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

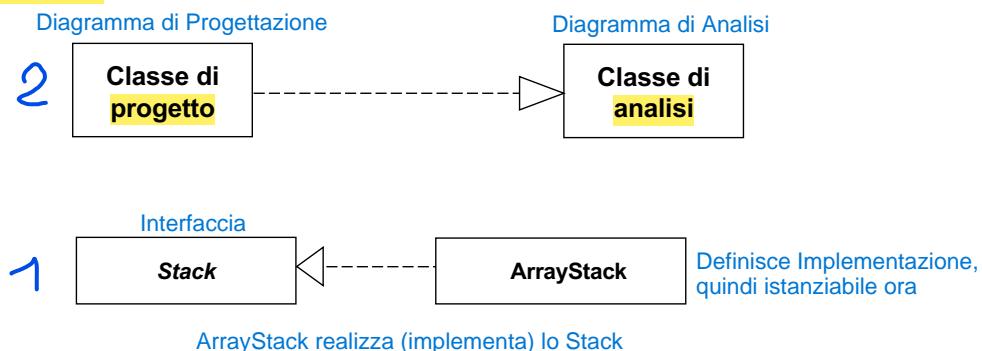


60

# Raffinamento

NON LO USEREMO  
NEGLI ESERCIZI

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

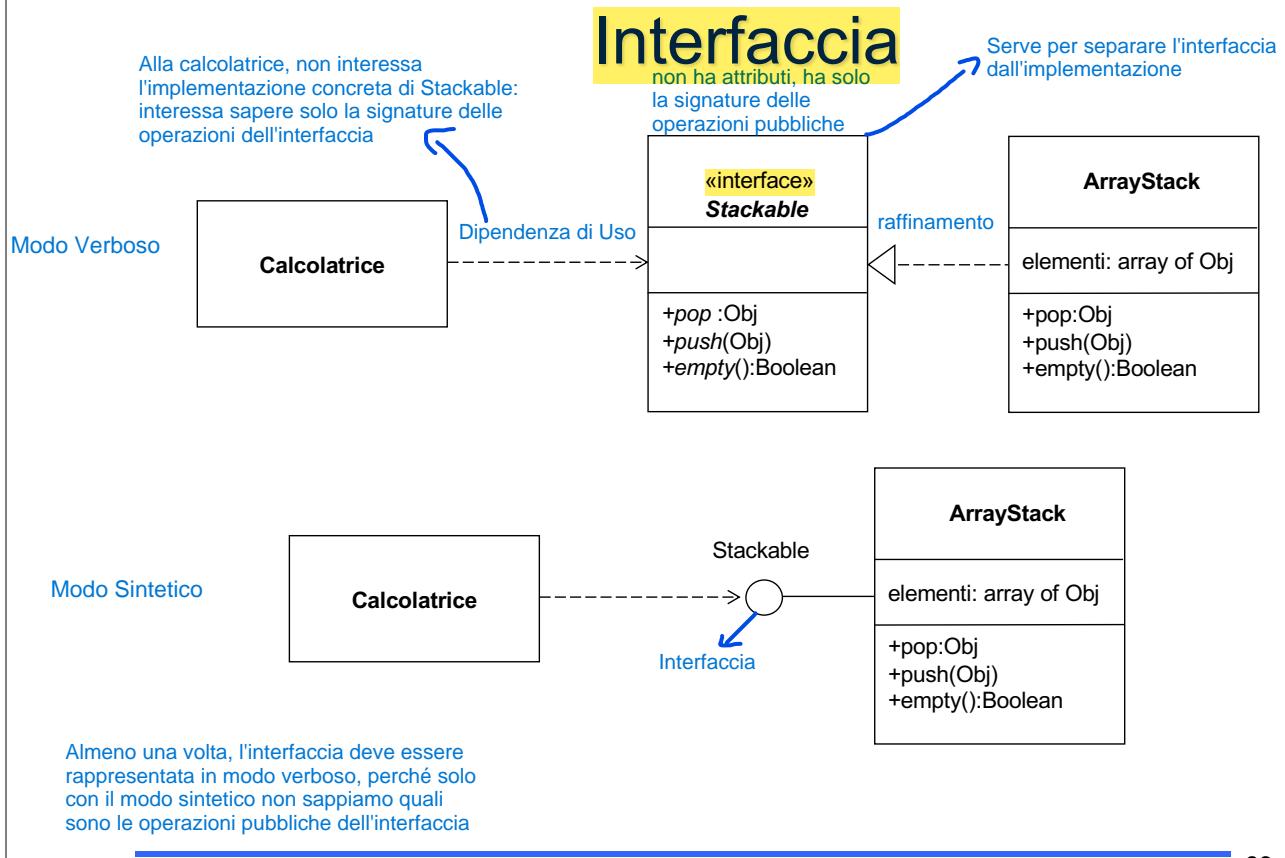


61

# Interfaccia

- Una **interfaccia** è un insieme di funzionalità pubbliche identificate da un nome (non istanziabile, bisogna realizzare l'implementazione delle funzionalità pubbliche)
- 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»

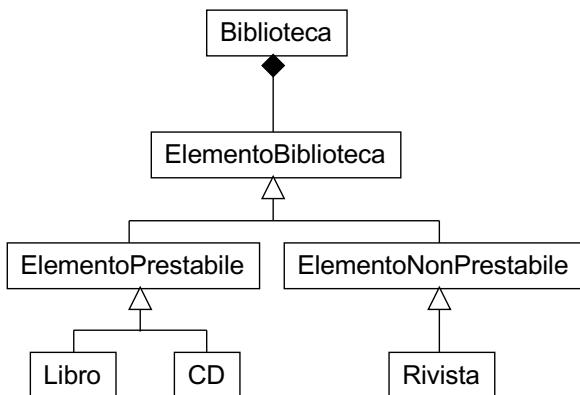
62



## Interfaccia vs. ereditarietà

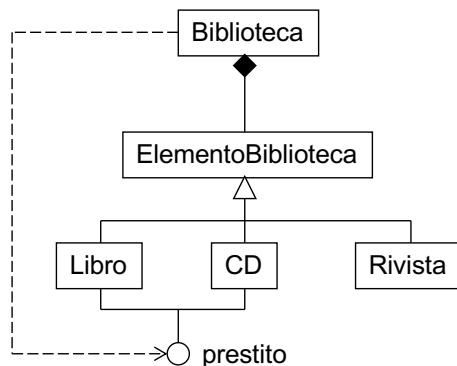
Entrambi Diagrammi di Analisi

Gerarchia di Specializzazione Profonda  
(cattiva forma di modellazione)



Non usare  
specializzazione  
per effetti transitori  
(non permanenti)

Uso di Interfacce  
(buona forma di modellazione)



# Analisi vs. progettazione

Nei COMPITI 90%  
Diagrammi di Analisi

- **Classi di analisi** rappresentano il punto di partenza per comprendere e modellare il dominio del problema.
    - ⇒ rappresentano un'astrazione nel dominio del problema
    - ⇒ corrispondono chiaramente a concetti concreti del mondo del business Ogni classe di analisi è chiaramente collegata a un concetto che gli utenti del sistema conoscono o utilizzano.
    - ⇒ escludono tutti i dettagli implementativi Si concentrano su cosa deve fare il sistema, non su come farlo.
    - ⇒ hanno un insieme ridotto, coeso e ben definito di responsabilità (operazioni che saranno fatte)
    - ⇒ indicano gli attributi che saranno *probabilmente* inclusi nelle classi di progettazione Gli attributi rappresentano informazioni fondamentali
    - ⇒ le loro operazioni specificano i principali servizi offerti dalla classe (per definire le operazioni che saranno fatte) (Le operazioni definiscono servizi generali che il sistema deve offrire, ma non entrano nei dettagli)
  - **Classi di progettazione** Le classi di progettazione si basano sulle classi di analisi, ma sono un livello successivo nel processo di sviluppo.
    - ⇒ le loro specifiche sono complete per cui possono essere direttamente implementate (ci devono essere tutti gli attributi e operazioni con parametri giusti)
    - ⇒ nascono dal dominio del problema per raffinamento delle classi di analisi, oppure dal dominio della soluzione
- Non devono esserci sovrapposizioni o confusione sui ruoli delle classi.
- Le classi di progettazione includono tutte le informazioni necessarie per essere tradotte in codice.
- Alcune classi derivano direttamente dal dominio del problema.  
Altre classi sono introdotte per supportare la soluzione tecnica.

65

I Metodi non vengono da UML, ma da OMT (perché UML definisce solo la Sintassi e non un Metodo, che fornisce OMT)

## 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”

Ogni classe ha un suo piccolo insieme di responsabilità

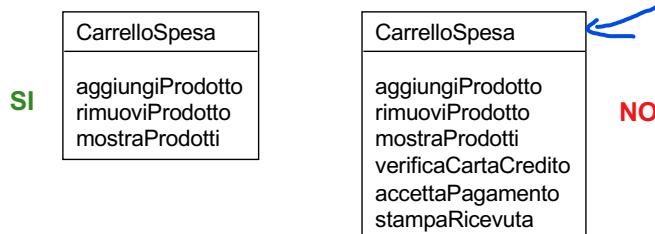
Quando disegnate una classe, chiedetevi sempre se avete chiaro le istanze della classe



66

## Identificare le classi d'analisi

- ⇒ Una classe è associata a un piccolo e ben definito insieme di responsabilità (normalmente tra 3 e 5) (non mischiare altri TIPI di operazioni insieme tra loro)

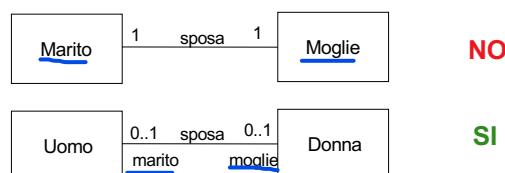


- ⇒ Nessuna classe può essere isolata (Classi devono avere relazioni, se no togliere)  
⇒ Evitare di avere poche classi troppo complesse, ma anche tante classi troppo semplici (La granularità viene definita dalle specifiche)

67

## Identificare le classi d'analisi

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

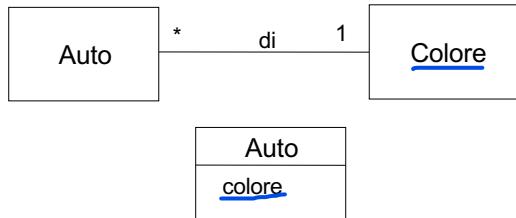


- ⇒ Evitare le gerarchie di specializzazione profonde

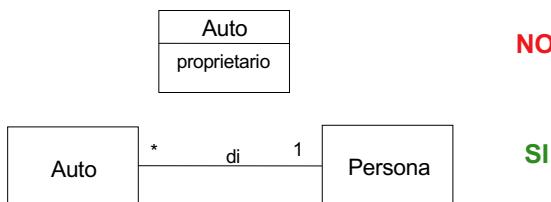
68

## 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

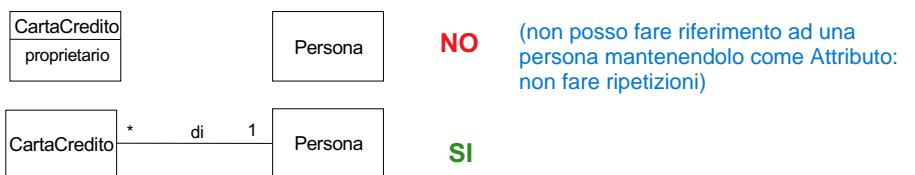


69

## 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  
(Occhio ai Cicli)

70

# Identificare le associazioni d'analisi

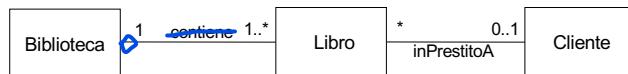
ASSOCIAZIONI DEVONO ESPRIMERE UNA SPECIFICA DI TIPO STRUTTURALE, NON UNA FUNZIONALITÀ

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

TUTTE LE ASSOCIAZIONI DEVONO AVERE UN NOME



**NO** (Va nei Diagrammi di Casi d'Uso, questo esempio)



**preferibile**

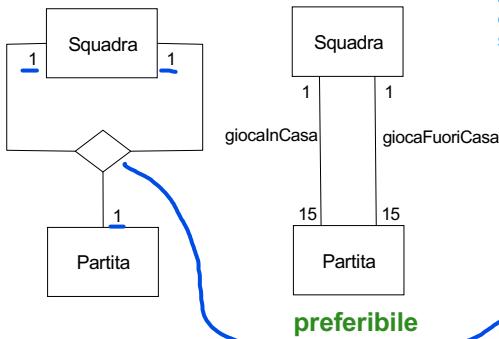
- Molte associazioni ternarie possono essere scomposte in due associazioni binarie

Quando una delle classi coinvolte in una relazione naria ha massima cardinalità 1, significa che ogni combinazione delle altre classi è associata a una singola istanza della classe con cardinalità massima 1.

Le molteplicità scritte sono le molteplicità di Ritorno

ES: (Una squadra + Una squadra) ha 1 Partita

Partita ha molteplicità max 1 perché è identificata da una Coppia Univoca Squadra-Squadra, quindi Falsa Ternaria



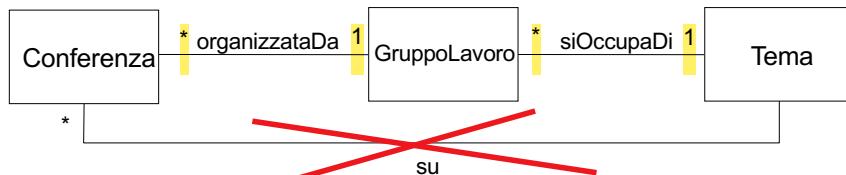
Le associazioni ternarie (o più in generale le associazioni n-arie) esprimono vincoli tra le classi che partecipano all'associazione. Questi vincoli definiscono come gli oggetti delle classi coinvolte possono essere correlati tra loro.

71

# Identificare le associazioni d'analisi

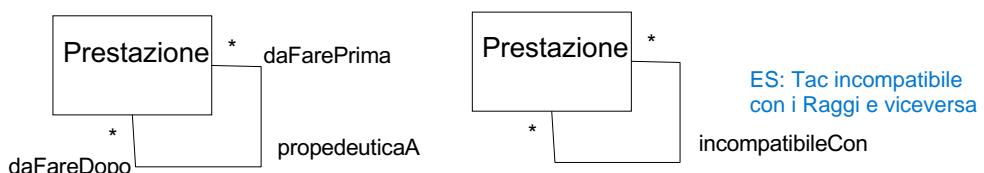
Associazioni Ridondanti: o la eliminate o la lasciate e metti uno slash (ES: /su)

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



- Quando appropriato, specificare i ruoli (Quando Associazione ad Anello non Simmetrica)

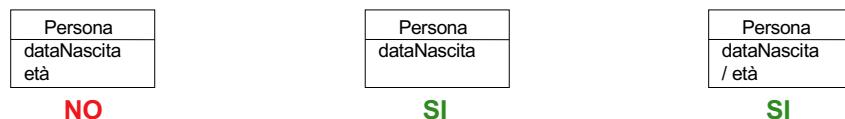
Non Simmetrica, quando hanno Molteplicità Diverse



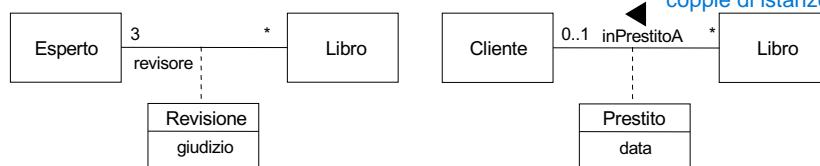
72

## 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 con uno Slash



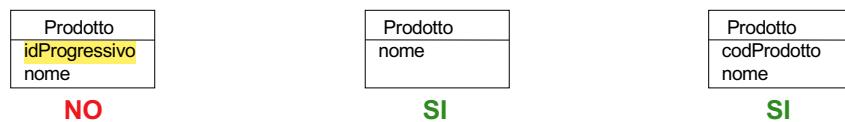
- ⇒ Se una proprietà dipende dalla presenza di un'associazione, rappresentarla con un attributo dell'associazione (usare Classe Associativa, però scatta la semantica di Unicità (non possibile copie di istanze associative duplicate))



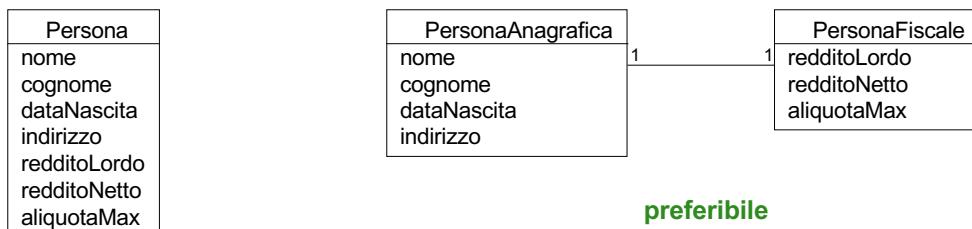
73

## 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



74

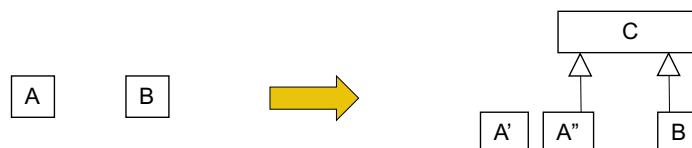
## Raffinamenti

- 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 (Aggiungere sottoclassi per rendere obbligatorie delle associazioni)



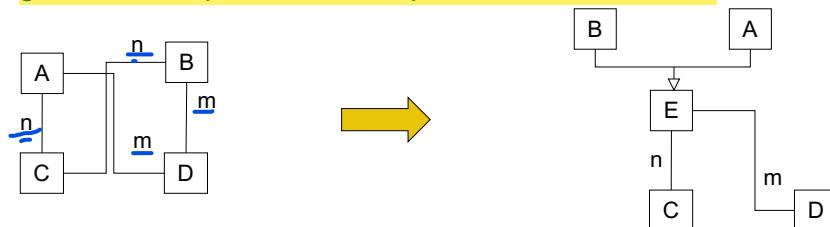
↳ In caso di difficoltà nel generalizzare, forse una classe sta giocando due ruoli differenti: può convenire spezzarla in due classi



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

Raffinamento delle Classi e Associazioni (Per Passaggio da Diagramma di Analisi a Diagramma di Progettazione)

- Con le classi di progettazione si specifica esattamente come le classi assolveranno le loro responsabilità → cioè, vengono definite in modo preciso le operazioni di ogni classe all'interno di un sistema software
- Ciascuna classe deve essere:
  - ⇒ completa, ossia fornire ai suoi clienti tutti i servizi che essi si aspettano (Ogni classe deve avere tutte le operazioni necessarie per soddisfare le aspettative dei suoi utilizzatori)
  - ⇒ 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 (eliminando qualsiasi ridondanza o complessità inutile)
  - ⇒ massimamente coesa, ossia modellare un unico concetto astratto (il che significa che tutti i suoi metodi e attributi devono essere correlati a un singolo ruolo o responsabilità)
  - ⇒ minimamente interdipendente, ossia essere associata all'insieme minimo di classi che le consente di realizzare le proprie responsabilità (La classe dovrebbe avere il minor numero possibile di dipendenze da altre classi)

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 (il verso dell'associazione) → Il Raffinamento dell'Associazione si occupa di definire il verso dell'Associazione (dato che nel Diagramma di Analisi sono Bi Direzionali, bisogna cambiarle in Mono Direzionali)
  - ⇒ la molteplicità a entrambi gli estremi
  - ⇒ il nome del ruolo destinazione

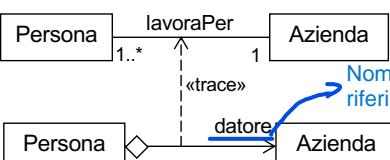
78

MOLTEPLICITÀ  
RIMANGONO LE  
STESSE

# Identificare le associazioni di progettazione

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

Diagr.Analisi



raffinamento

Diagr.Prog

Persona contiene un attributo che ha un riferimento all'azienda a cui lavora

Il Carico di Lavoro predominante guida il Raffinamento in questo verso: se la maggior parte delle operazioni sono su Persona

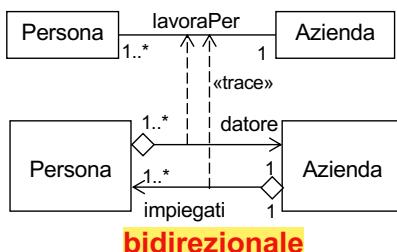
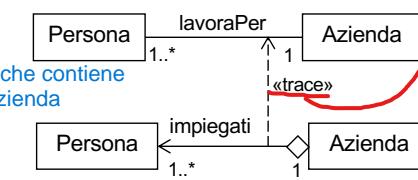
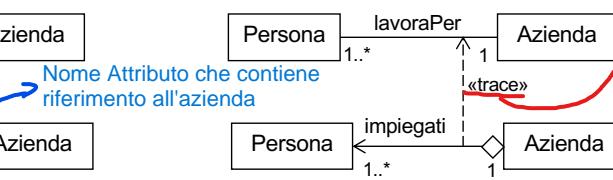
Nome Attributo che contiene riferimento all'azienda

monodirezionali

TUTTE E TRE I RAFFINAMENTI SONO COMPLETI E NON RIDONDANTI (TRANNE LA TERZA CHE È RIDONDANTE, MA ACCETTABILE)

Noi non lo utilizzeremo perché faremo diagrammi separati

Carico di Lavoro al contrario rispetto a sinistra



bidirezionale

Questo è il caso in cui Carico di Lavoro è Bilanciato. È Ridondante ma migliora le prestazioni. Inoltre, la ridondanza mi preoccupa di meno rispetto al ER perché tramite Incapsulamento non ho inconsistenze (aggiornamenti dei campi incapsulati tramite Metodi che non creano incosistenze)

79

Se non si è sicuri, mettere rombi bianchi

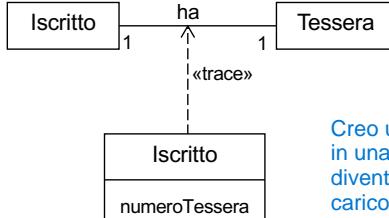
# Identificare le associazioni di progettazione

## Associazioni uno-a-uno (due modi)

In Iscritto, per tessera non ho un riferimento, bensì un valore (perché è una Composizione)



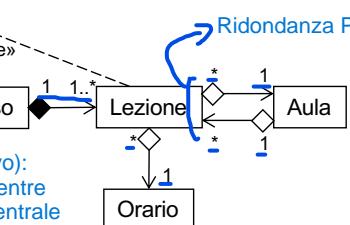
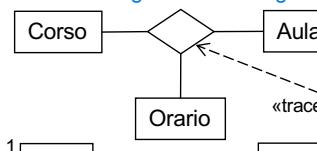
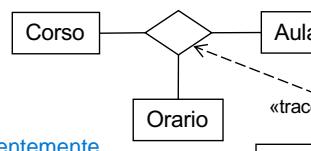
1) Indipendentemente dalla presenza o meno di una classe associativa, bisogna creare una quarta classe (che reifica l'associazione)



Creo una Classe, inglobando le due in una, decidendo quale dei due diventa attributo e classe (in base al carico di lavoro), unendo gli attributi

## Associazioni ternarie o n-arie

Le molteplicità delle ternarie sono quelle di ritorno. Quelle di Andata sono tutte ad \* (Diagramma di Analisi). Nel Diagramma di Progettazione sono quelle di andata.



Molteplicità (sono quelle di Arrivo): Dalla Centrale alle altre \* a 1 mentre Dalla Classe Selezionata alla centrale 1 a 1..\*

Se c'è la Classe Associativa, ne basta 1 (cioè uso la Classe Associativa come quarta classe)

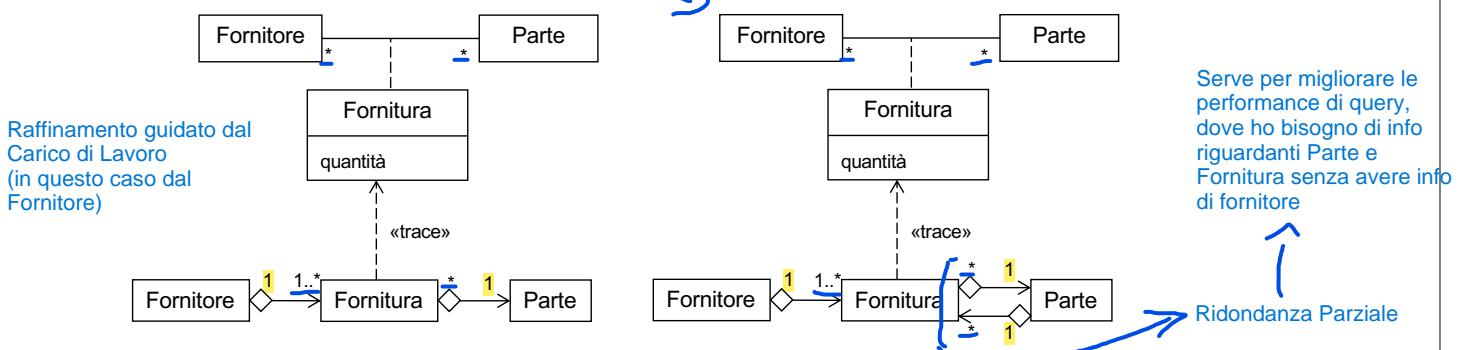
2) Quando si progetta un n-aria, in base al carico di lavoro, si mette il verso da una classe alla classe centrale, poi verso le altre (cioè scelgo la classe più importante)

La classe Reificata contiene i tre OID, quindi le classi esterne devono avere molteplicità ad 1

80

# Identificare le associazioni di progettazione

## □ Classi associative (due modi)



La Classe Associativa viene raffinata in una Classe. Le Associazioni passano SEMPRE per la Classe Associativa raffinata  
Molteplicità: sui lati interni copio le Molteplicità, sui lati esterni metto 1 (perché la classe associativa contiene gli OID in modo univoco)

Nel caso di Ternarie, non si aggiunge una Classe, ma si usa la Classe Associativa raffinata

81

**5**

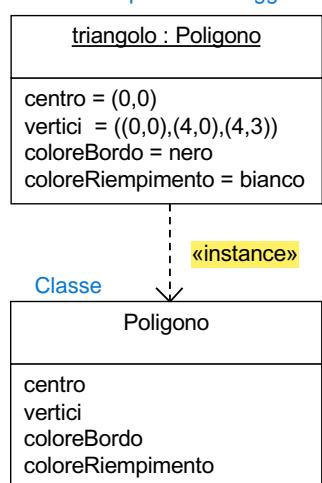
NON VERRÀ  
CHIESTO DI  
DISEGNARLO  
ALL'ESAME

## Diagrammi degli oggetti

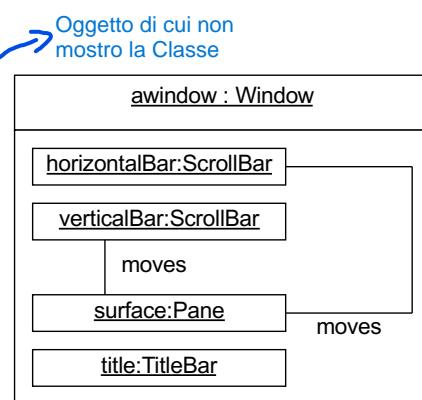
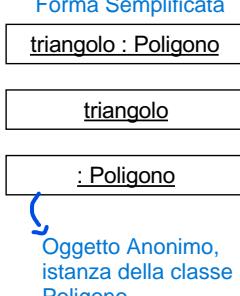
(Istanza del Diagramma delle Classi)

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

Notazione Completa di un Oggetto



Forma Semplificata



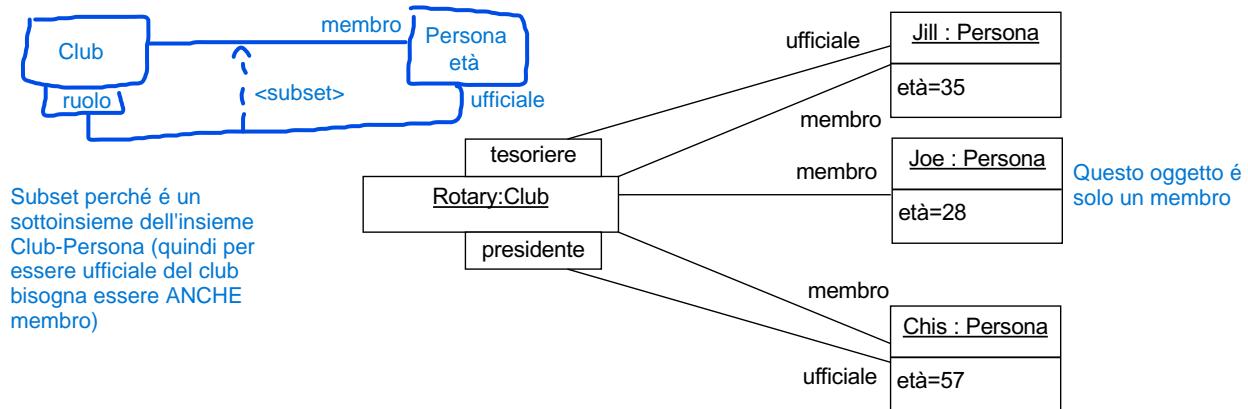
82

# Diagrammi degli oggetti

NON USATO TANTO DI FREQUENTE, MA UTILE PER MOSTRARE COME INSTANZIARE LE CLASSI

- 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

Diagramma Classi



83

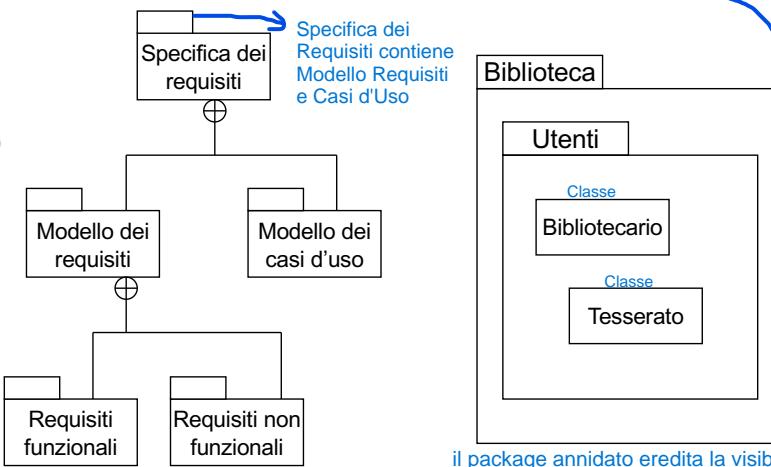
## 6

NON VERRÀ CHIESTO DI DISEGNARLO ALL'ESAME

## Diagrammi dei package

- Un package è un raggruppamento di elementi del modello semanticamente correlati (cioè, dal punto di vista del significato, sono collegati tra loro)
- 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

- Il package annidato "Utenti" vede lo spazio dei nomi di "Biblioteca". Tutto ciò che è dichiarato nel package "Biblioteca" (es. classi o elementi) è visibile al package "Utenti" e alle classi al suo interno, cioè a "Bibliotecario" e "Tesserato".  
- Il contrario non è vero: Il package "Biblioteca" non vede automaticamente gli elementi dichiarati dentro "Utenti". Pertanto, la classe "Bibliotecario" o "Tesserato" non è visibile direttamente a elementi dichiarati in "Biblioteca", a meno che non ci sia un'importazione esplicita o un riferimento chiaro.



Il package annidato eredita la visibilità dello spazio dei nomi del package che lo contiene, propagandosi verso il basso.

I package annidati possono vedere lo spazio dei nomi dei package che li contengono. Questo significa che le classi definite nel package esterno sono accessibili nel package annidato. Il package esterno non ha accesso agli elementi definiti nel package annidato, a meno che non siano esplicitamente resi accessibili

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 (Un Package usa un elemento di un altro Package)
- «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

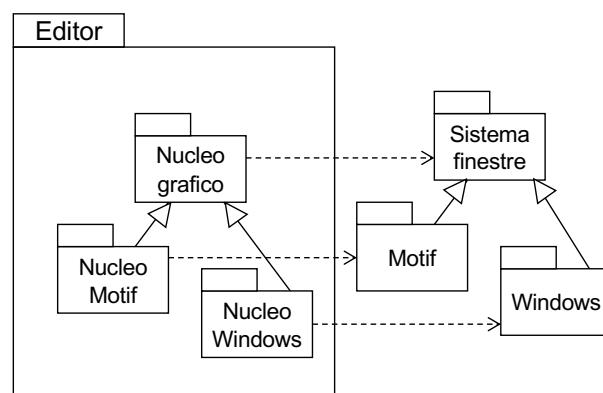


85

## Diagrammi dei package

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

(Un Package può avere delle Specializzazioni: si può creare un Package a livello Astratto)



86

In pratica, vengono usati spesso come Diag. Architetturali nelle fasi preliminari di Progettazione al posto del Diag. Componenti per ridurre complessità e dipendenze (poi uso Diag. Componenti)

## Individuare i package d'analisi

In UML, non è definito cosa "mettere" nei Package. Ci sono solo delle Linee Guida (Creazione Package basandosi su Associaz. PartOf e/o Generalizzazioni dei Diag. Classi)

- 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 (Per Migliorare la Manutenibilità del Software, cioè ridurre le dipendenze)

87

7

(Aspetti Dinamici e Funzionali)

## Diagrammi di interazione

Essi descrivono il flusso di messaggi tra gli oggetti e come questi si coordinano per realizzare una determinata funzionalità

- Rappresentano la struttura dell'interazione tra oggetti durante uno scenario (Usato per modellare Sequenze di Messaggi che vengono scambiati tra Oggetti)
- Esistono quattro tipi di diagrammi di interazione, ognuno rivolto a un particolare aspetto:

Useremo questo

- ⇒ **Diagramma di sequenza**: enfatizza la sequenza temporale degli scambi di messaggi (enfatizza l'ordine temporale con cui i messaggi vengono scambiati tra gli oggetti)
- ⇒ **Diagramma di comunicazione**: enfatizza le relazioni strutturali tra gli oggetti che interagiscono (enfatizza chi parla con chi)
- ⇒ **Diagramma di sintesi dell'interazione**: illustra come un comportamento complesso viene realizzato da un insieme di interazioni più semplici (permette di rappresentare interazioni complesse attraverso la sintesi di più diagrammi di interazione più semplici)
- ⇒ **Diagramma di temporizzazione**: enfatizza gli aspetti real-time di un'interazione (enfatizza messaggi sincroni-asincroni)

88

Scambio di un Messaggio =  
Oggetto invoca un Operazione  
di un altro Oggetto

Un'interazione è considerata "un'unità di comportamento di un classificatore"  
perché rappresenta come un oggetto (il classificatore) si comporta in risposta  
agli scambi di messaggi con altri oggetti all'interno di un sistema.

È una sequenza di messaggi  
scambiati tra oggetti che seguono  
un ordine temporale per eseguire  
una certa funzionalità.

## Terminologia

I classificatori sono utilizzati per rappresentare gli elementi che partecipano alla sequenza di interazioni in un sistema. Ogni linea di vita in un diagramma di sequenza è collegata a un classificatore. La linea di vita rappresenta l'istanza di un classificatore.

Il contesto è lo scenario o la  
situazione specifica  
in cui l'interazione  
si svolge

- 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 (Ambito in cui disegniamo il Diagramma di Sequenza)
- 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

Diagramma di  
Sequenza è  
molto utilizzato  
per  
documentare  
un Caso d'Uso

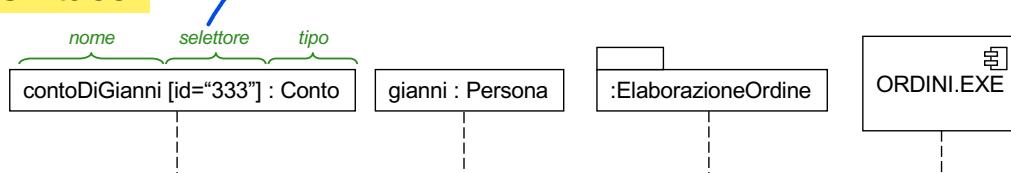
(Vengono scambiati messaggi tra due linee di vita per far eseguire un'attività)

89

Linea Tratteggiata: Evoluzione  
del tempo e che cosa succede  
durante la vita del Classificatore

### Sintassi:

Mi va a selezionare quella Istanza di Classe che ha come attributo id = 333



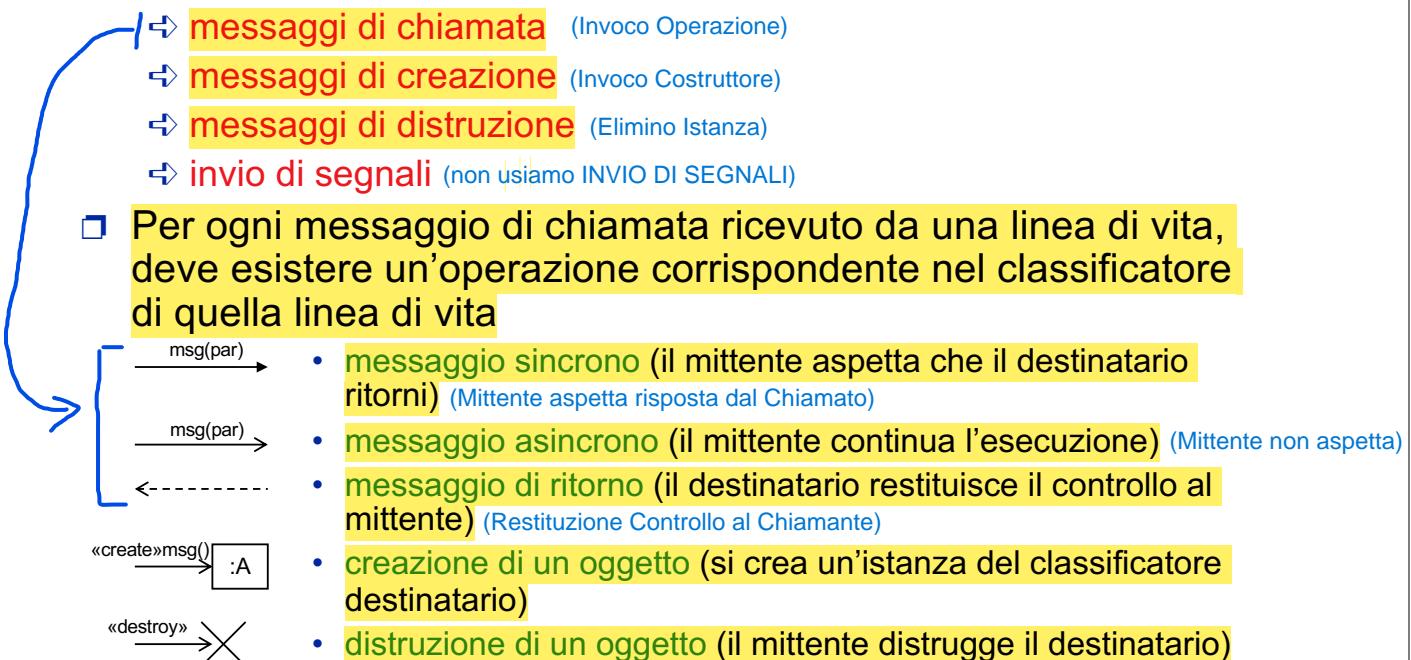
Istanza di una  
Classe= Nome  
Oggetto +  
Selettore : Classe

- 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

Ad esempio, in un sistema di acquisto online, una linea di vita potrebbe rappresentare il ruolo di "cliente" senza riferirsi a un cliente specifico. Questo permette di descrivere come qualsiasi cliente interagisce con il sistema, indipendentemente dalla sua identità.

90

# Messaggi



91

NON È UN DIAGRAMMA DI TIPO STATICO, MA DI TIPO DINAMICO-FUNZIONALE

# Diagrammi di sequenza

Lettura Diagramma  
dall'alto verso il basso

- 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

Un'attivazione rappresenta il periodo durante il quale una linea di vita esegue un'azione. Questa azione può essere:

- Diretta: L'oggetto/attore esegue l'operazione autonomamente.

- Indiretta: L'oggetto/attore delega l'operazione chiamando un'altra procedura, funzione o servizio.

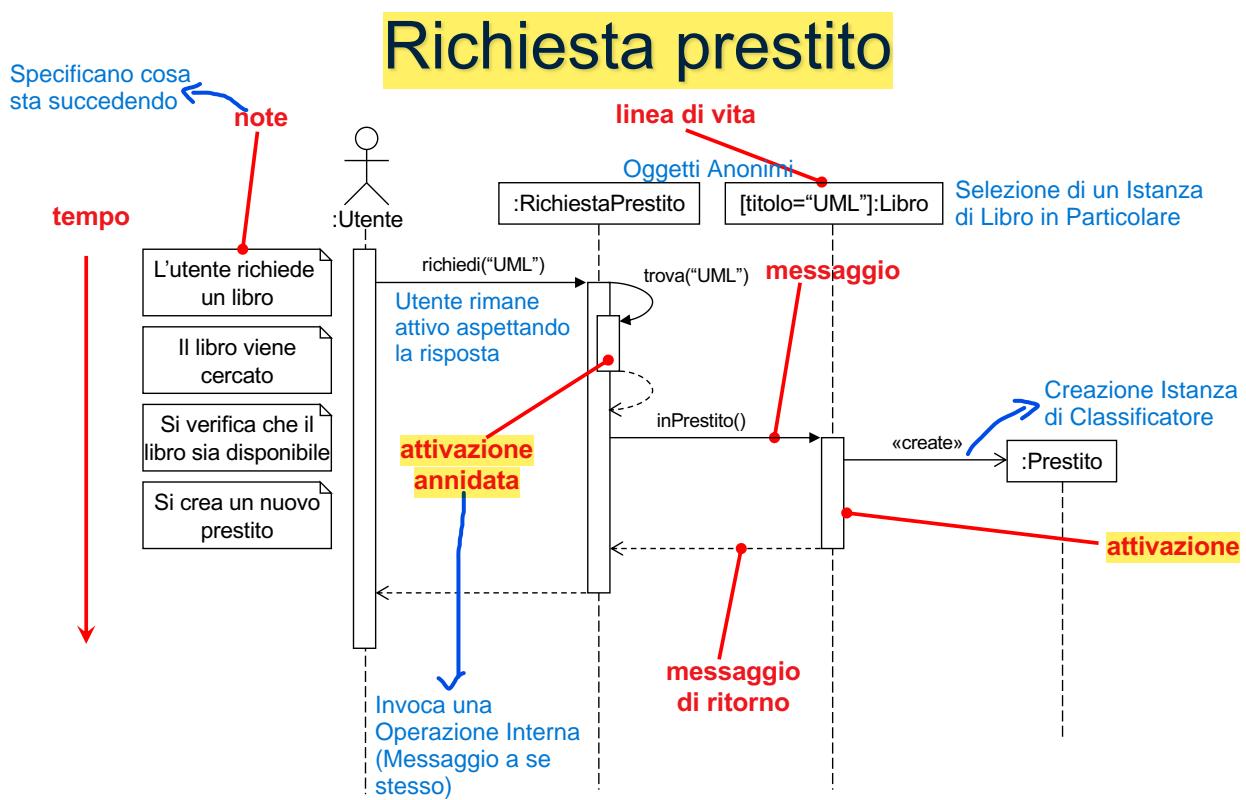
↳ Hanno due dimensioni: la dimensione verticale rappresenta il tempo mentre quella orizzontale rappresenta le linee di vita

↳ 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

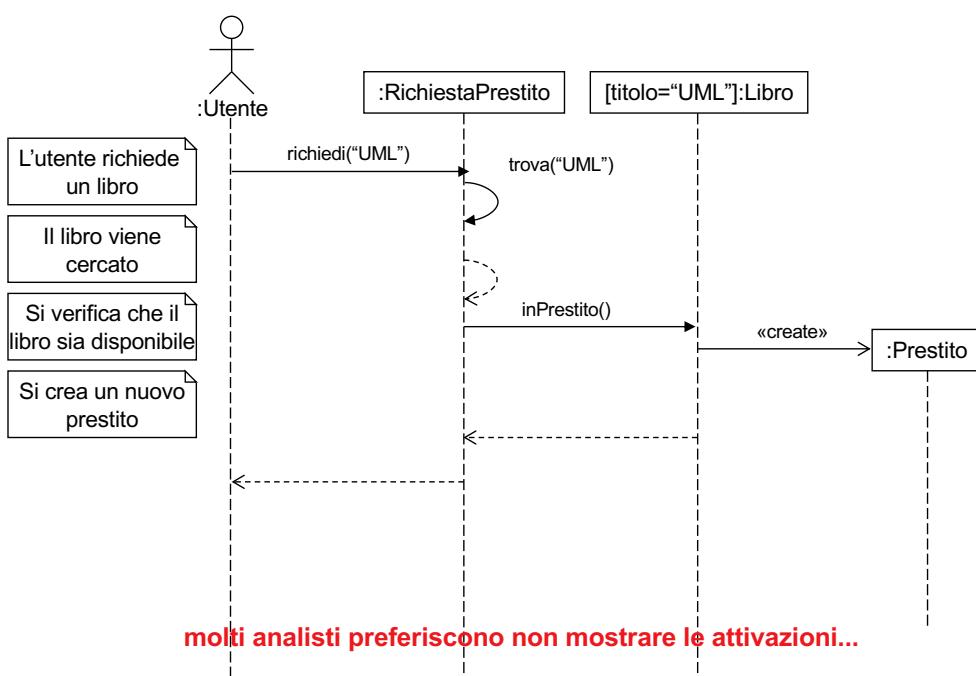
92



93

MOSTRARE SEMPRE LE ATTIVAZIONI

## Richiesta prestito



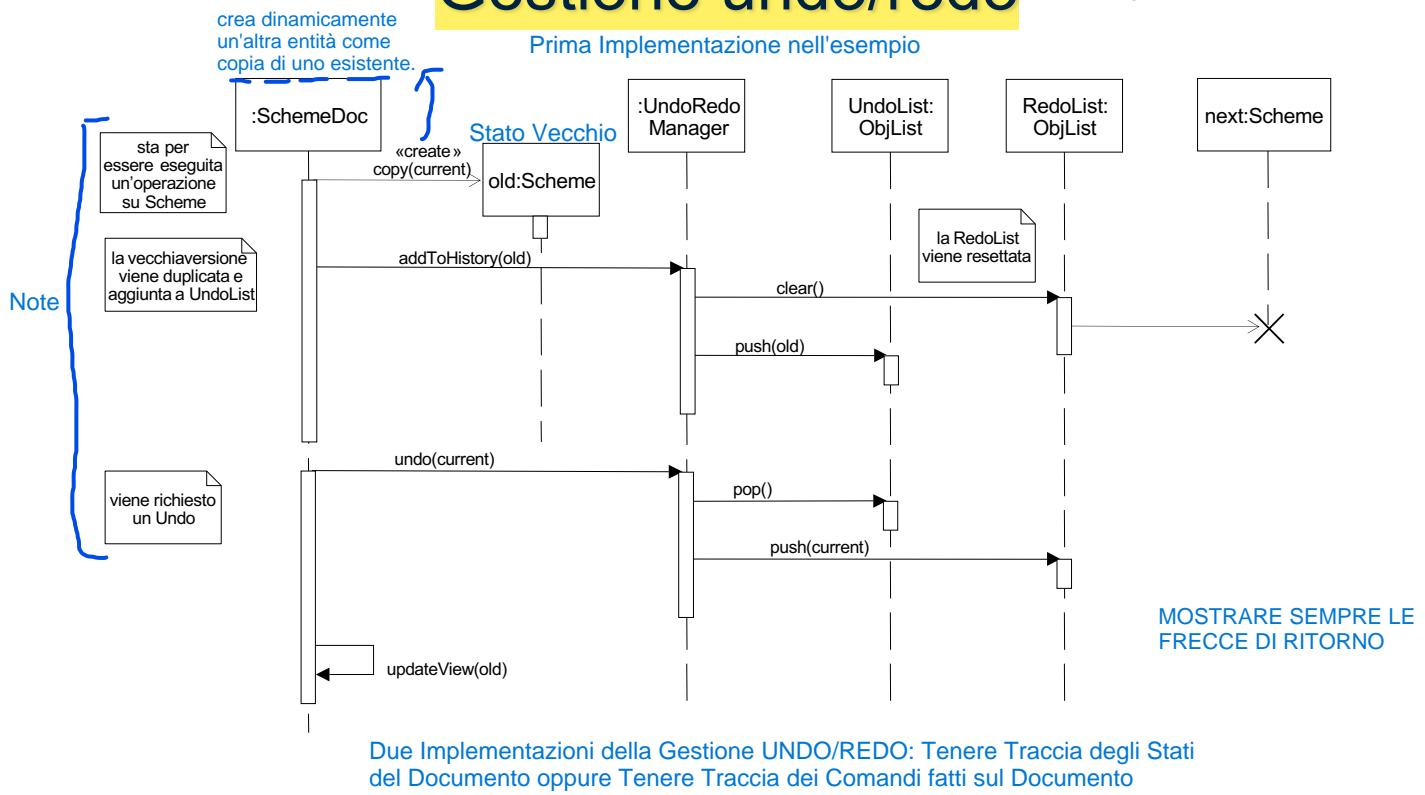
94

UNDO: Annullare gli Effetti dell'Ultima Operazione Eseguita

## Gestione undo/redo

REDO: Cancellare gli Effetti dell'UNDO

Prima Implementazione nell'esempio



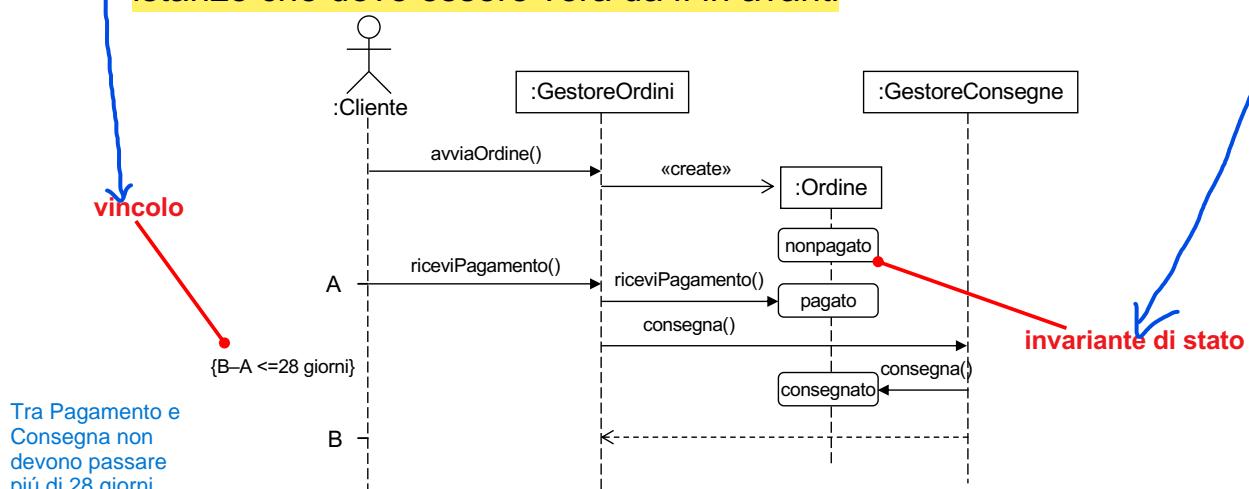
95

IL DIAGRAMMA DI SEQUENZA  
NON MOSTRA TUTTE LE COSE  
CHE POSSONO SUCCEDERE  
(VENGONO MOSTRATE SOLO LE  
PRINCIPALI VARIANTI/SCENARI)

## Invarianti di stato e vincoli

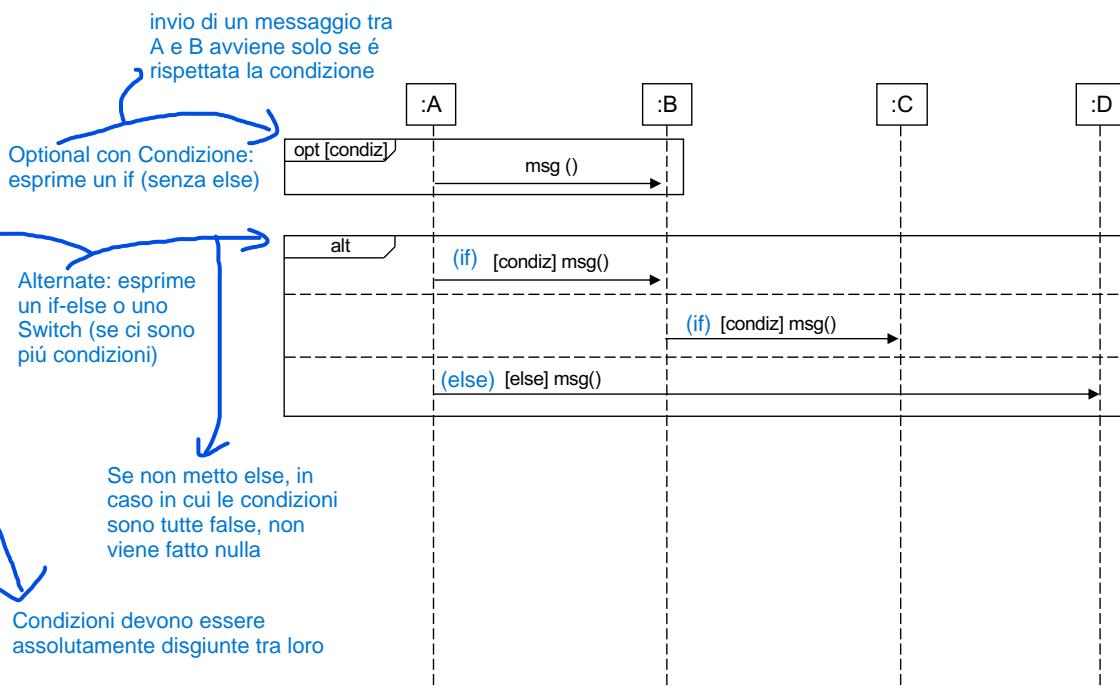
NON USEREMO GLI STATI  
IN QUESTO DIAGRAMMA

- ☐ 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



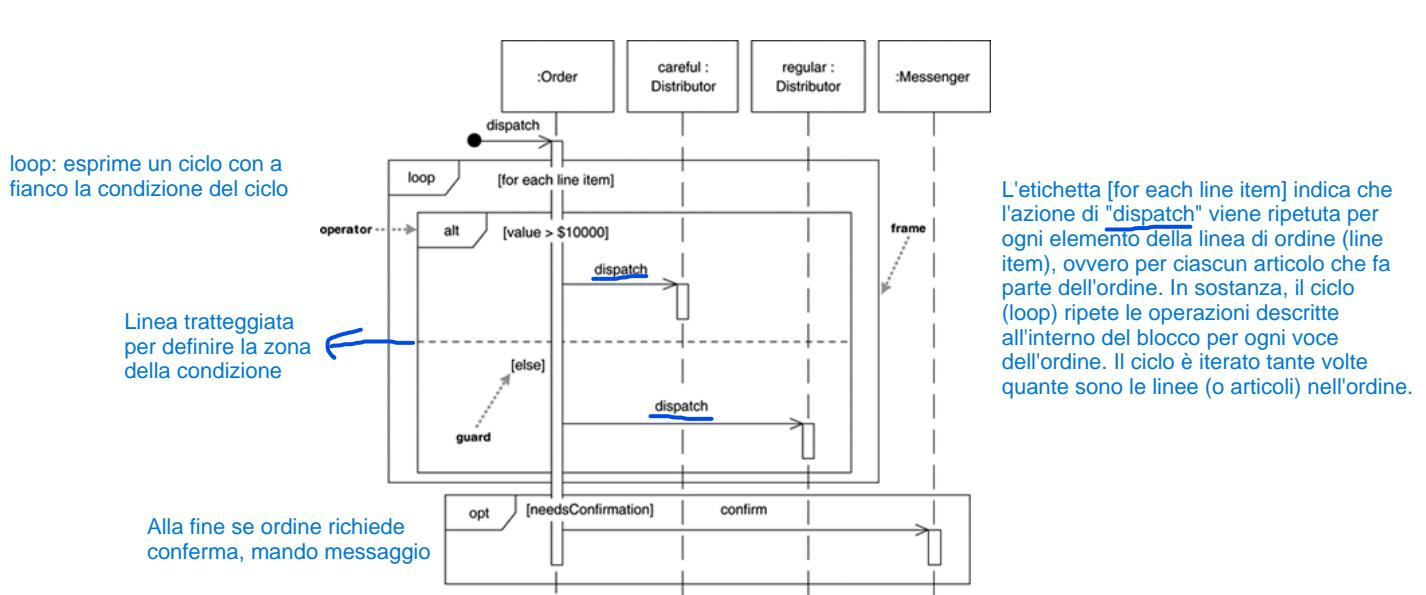
96

## Frammenti combinati



97

## Frammenti combinati



98

# 8

Diagramma di tipo Dinamico

## Diagrammi di stato

(Rappresentare un Comportamento Dinamico di un'istanza di un Classificatore)

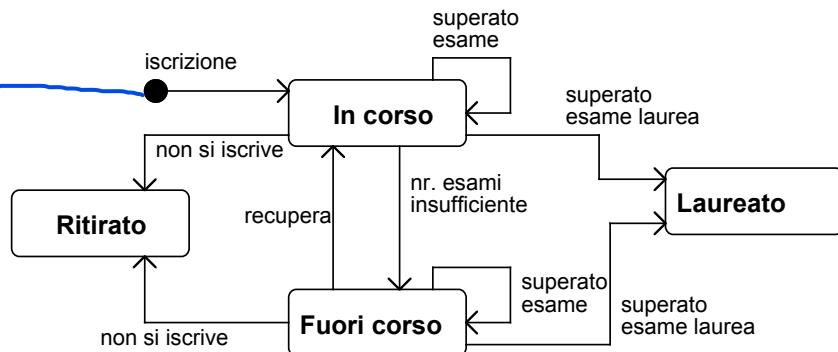
(esaurito = completo)

- I diagrammi di stato descrivono in modo esauritivo l'evoluzione temporale delle istanze di un classificatore (classe, caso d'uso, sottosistema) in risposta alle interazioni con altri oggetti (e uno solo)
- Ogni classe può avere associato un diagramma di stato

Harel è una  
estensione degli  
"automi a stati  
finiti"

→ 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.

Punto in cui si entra  
nel diagramma:  
spesso indica la  
creazione  
dell'istanza del  
classificatore



99

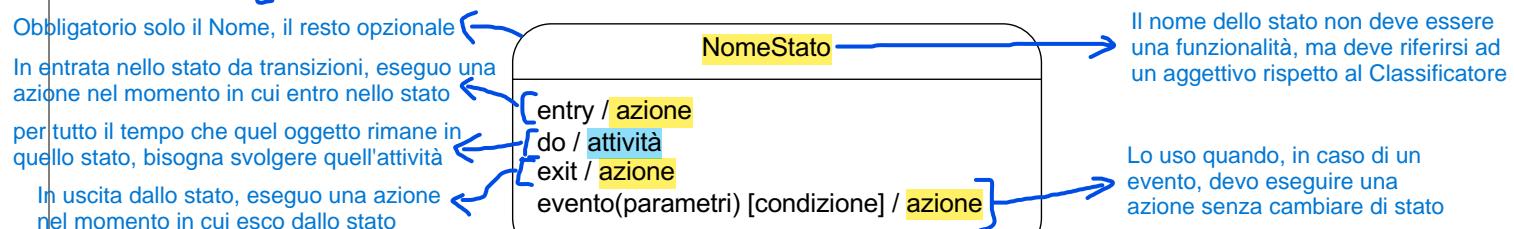
Eventi e Transizioni sono istantanee,  
mentre Stato occupa tempo

## Stati ed eventi

definita astrazione perché non tutti i cambiamenti di un attributo, porta ad un cambiamento di stato

- Lo **stato** di un oggetto in un certo istante è un astrazione dell'insieme dei valori dei suoi attributi e dei suoi collegamenti (associazioni)
  - ⇒ 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 (un evento è istantaneo, cioè atomico, non divisibile)
- 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

Si distinguono dal fatto  
che siano interrompibili  
o no, non dal tempo di  
esecuzione



entry: abbreviazione per evitare di scrivere l'azione in tutte le Transizioni in entrata  
exit: abbreviazione per evitare di scrivere l'azione in tutte le Transizioni in uscita  
evento...: abbreviazione per evitare di scrivere l'azione in un auto-anello

100

AZIONI abbinate agli EVENTI (TRANSIZIONI),  
ATTIVITÀ abbinate agli STATI

# Transizioni

Regola ECA: QUANDO si verifica evento, SE condizione vera, ALLORA esegui azione e cambia di stato (Event Condition Action)

Sempre associate ad eventi

- 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 (atomico)
  - ⇒ 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 (Quando si verifica evento, se condizione vera, avviene transizione, se no, non avviene e rimane nello stato precedente)
- 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

Alla fine dell'attività (che termina e non ciclico), essa scatena un evento implicito e quindi Transizione



101

Come Disegnare Diagrammi Stati?  
1) Disegno Stati e Transizioni Principali (scenario base)  
2) Disegno le Varianti  
3) Ricontrollo per ogni stato se ci sono altre varianti

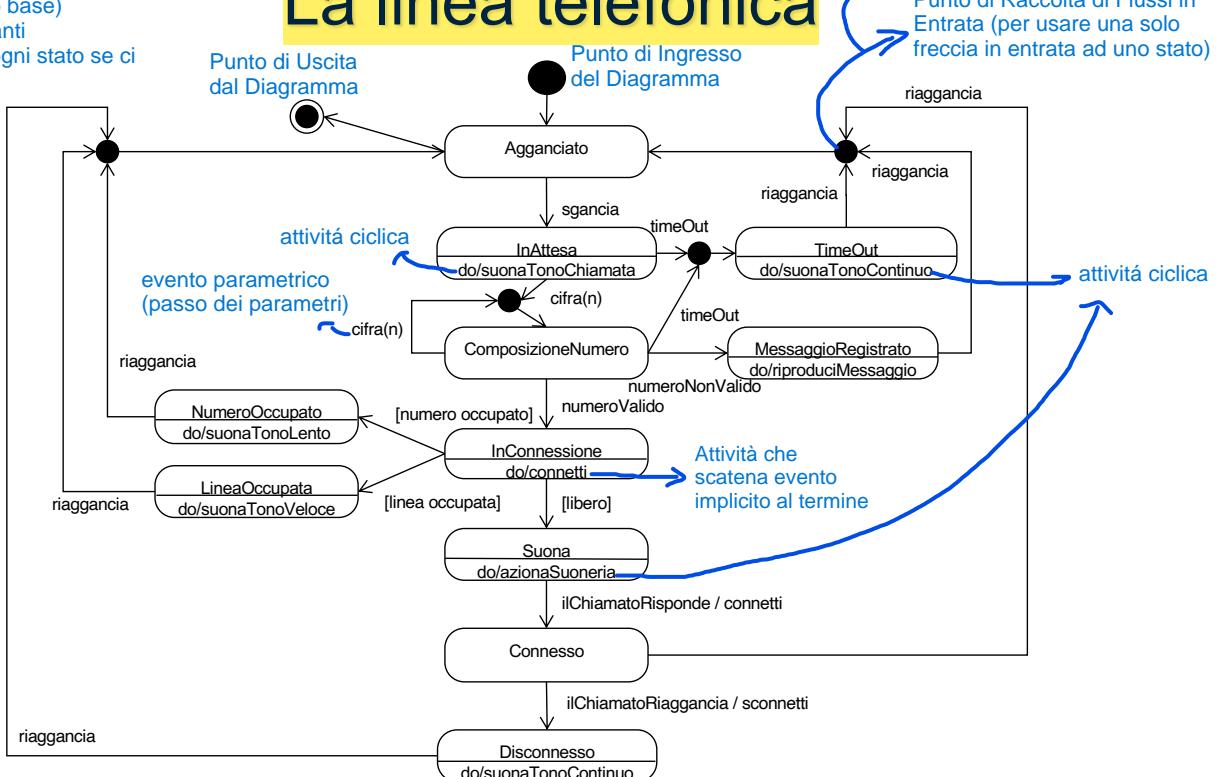
SI DISEGNANO PER RAFFINAMENTI SUCCESSIVI (DESIGN ITERATIVO), CIOÉ DA DIAGRAMMA SEMPLICE A COMPLESSO

# La linea telefonica

Punto di Uscita dal Diagramma

Punto di Ingresso del Diagramma

negli esercizi, si possono non usare Punto di Raccolta di Flussi in Entrata (per usare una solo freccia in entrata ad uno stato)



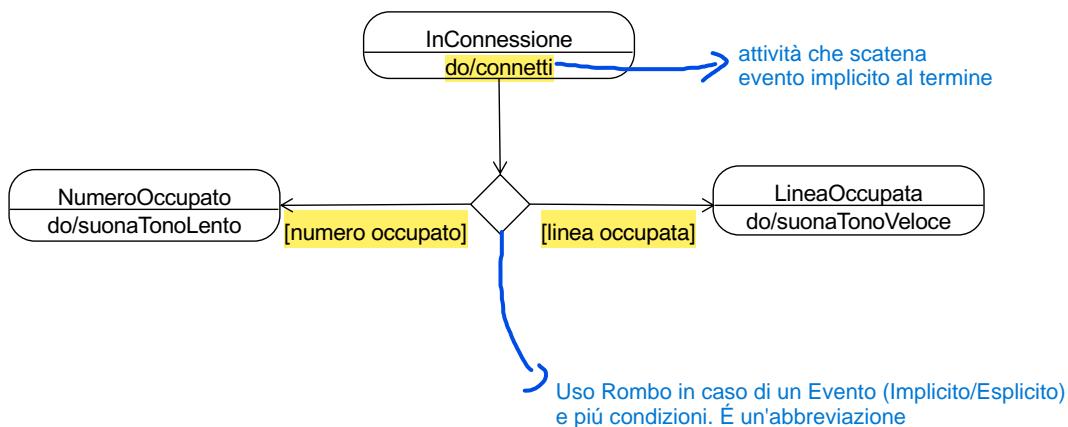
102

È rappresentato come un rombo

## Pseudo-stato di selezione

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

Un automa è un modello matematico usato per rappresentare un sistema che cambia il suo stato in risposta a determinati eventi o input, secondo regole predefinite.



103

Evento di Variazione VS Condizione:  
evento risponde alla domanda  
QUANDO, mentre condizione al SE

## Tipi di eventi

Però se ho Evento+Cond e scatta evento, ma Condizione Falsa, non avviene variazione e l'istanza rimane sullo stato precedente e la transizione avviene solo se l'evento si riverifica, non se la Condizione diventa Vera

- Evento di variazione:** si verifica nel momento in cui una condizione diventa vera
  - è denotato da un'espressione booleana, ad esempio: bilancio<0 (Condizione senza Parentesi)
  - può essere considerato come una condizione verificata continuamente sebbene in realtà verrà controllata solo al variare dei parametri coinvolti (Verificata Continuamente fino al momento in cui l'evento si scatena)
- Evento di segnale:** si verifica nel momento in cui un oggetto riceve un oggetto segnale 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 (non porsi il problema di specificare i parametri a livello di analisi)
  - 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 (momento esatto)
  - 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..." (Usare SINCE ad esempio, aspettare 10 secondi dopo l'entrata in uno stato diverso di quello attuale)

DOPO .. DA QUANDO .. (AFTER .. SINCE ..)

104

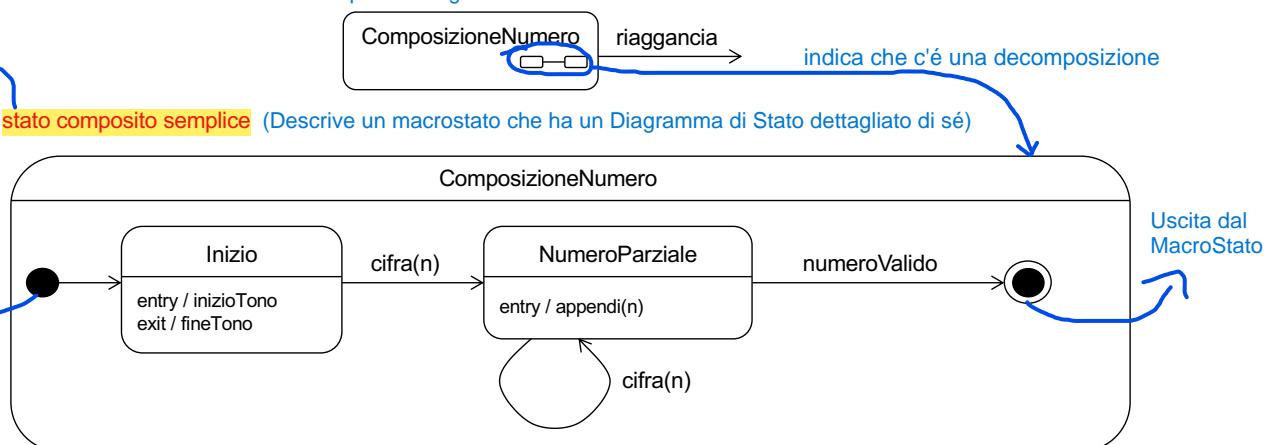
Un automa descrive il comportamento di un sistema o di un oggetto attraverso stati e transizioni che avvengono in risposta a eventi (Ciclo di vita che ha punti di ingresso e punti di uscita ben definiti). Automa = Diagramma degli Stati

## Stati compositi

- 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

Se un evento attiva una transizione nello stato composito, e quello stato è attivo, l'evento può attivare la transizione in qualsiasi punto all'interno degli stati annidati. In altre parole, gli stati annidati rispondono agli stessi eventi che attiverebbero una transizione nel macro-stato.

Utilizzo esclusivo per rendere una lettura più semplice

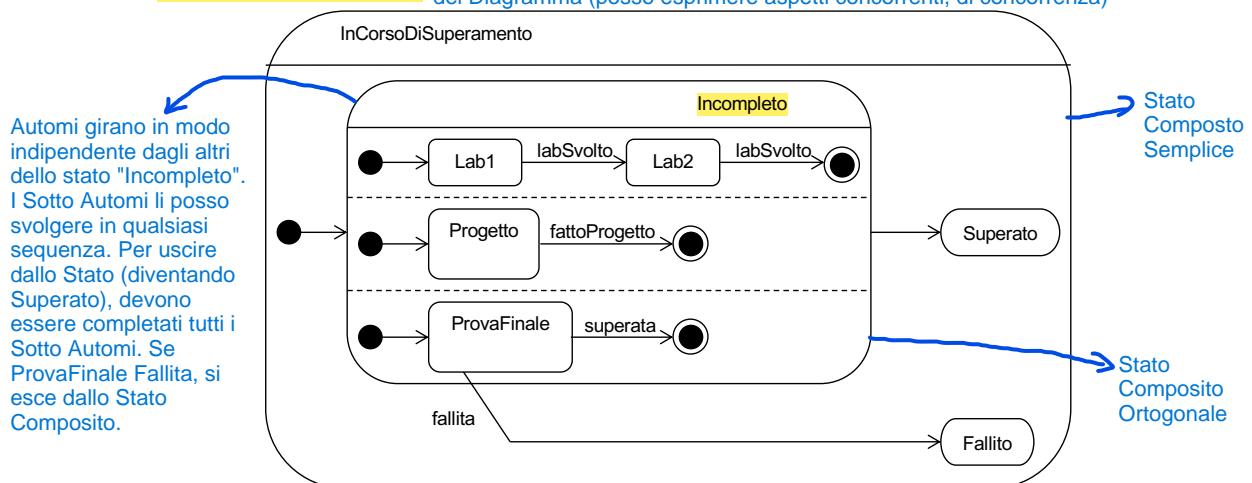


105

## Stati compositi

- Lo pseudo-stato finale di un automa viene applicato solo a quell'automa (cioè ogni sotto-automa ha il suo "occhio di bue", che termina SOLO il sotto-automa)

2 **stato composito ortogonale** Non è solo un modo per essere più chiari, ma sto cambiando espressività del Diagramma (posso esprimere aspetti concorrenti, di concorrenza)



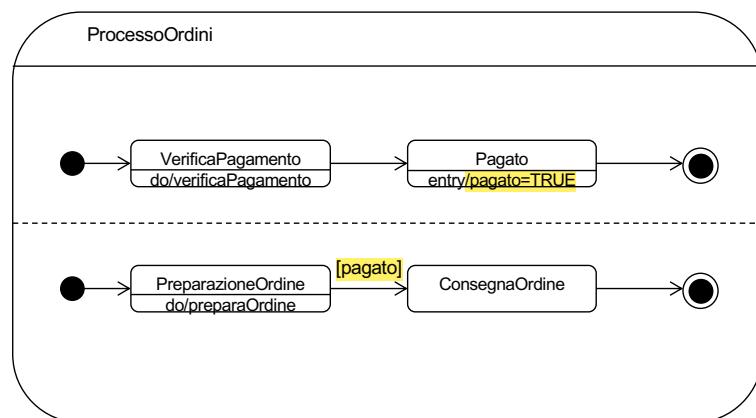
Si usa lo Stato Composito Ortogonale per diminuire la complessità del Diagramma (diminuire numero stati), perché senza di esso dovrei gestire tutte le combinazioni possibili

106

# Comunicazione tra automi

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

(In un Stato Composto Ortogonale, è possibile sincronizzare i Sotto Automi tra loro)



107

9

Modella Aspetti  
Dinamici e Funzionali

## Diagrammi di attività

(Permette di modellare concorrenza)

- Modellano un processo come un'attività costituita da un insieme di nodi connessi da archi (Racconta come si articola/funziona un Processo)
- 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

(Diagramma di Attività per raccontare come si svolge un Caso d'Uso, un'operazione etc.)

- Principalmente
- caso d'uso
  - operazione
  - classe
  - interfaccia
  - componente
  - collaborazione

Un workflow (flusso di lavoro) è una rappresentazione sequenziale delle attività, dei compiti o dei processi necessari per completare un obiettivo specifico. Si tratta di una serie di passaggi organizzati che definiscono come le attività devono essere eseguite, chi è responsabile di ciascuna fase, e in quale ordine devono essere completate.

108

ESEMPIO

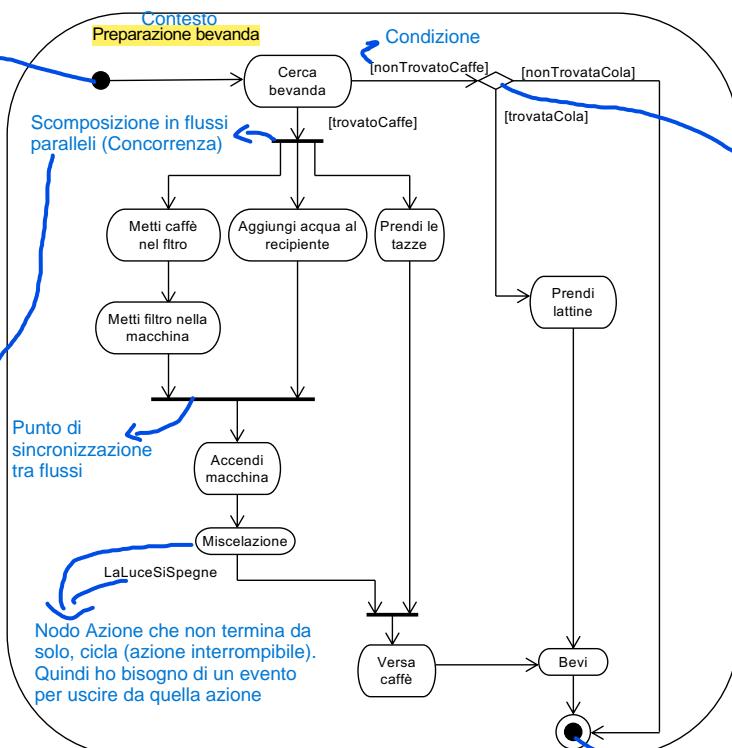
# Preparazione di una bevanda

Inizio Diagramma

Nel 90% dei casi, le frecce non hanno evento perché il nodo azione al termine scatena evento implicito (se è un evento che termina da solo: cioè cose da fare che terminano). Se è un nodo azione ciclico, quindi non termina da solo, bisogna specificare un evento esplicito nella freccia.

In questo diagramma, abbiamo quasi tutti nodi azione che terminano da soli. I nodi azione che scatenano evento implicito, sono atomici, non interrompibili.

Nella scomposizione in flussi paralleli, si può eseguire qualsiasi combinazione



Nodi = rappresentano azioni (cose che vengono fatte)  
 Frecce = rappresentano dei flussi di controllo (prima fai questo, poi fai questo)

Per esprimere un IF, è obbligatorio usare un rombo

109

Diagramma di Attività è un Grafo

# Attività

Sono modellate come reti di **nodi** connessi da **archi**

Categorie di nodi:

⇒ **nodi azione**, che rappresentano compiti atomici all'interno dell'attività (Il 90% dei nodi si interrompe da solo, il restante ha bisogno di un evento)

⇒ **nodi controllo**, che controllano il flusso all'interno dell'attività (rombo, ingresso e uscita diagramma etc..)

⇒ **nodi oggetto**, che rappresentano oggetti usati nell'attività

Categorie di archi:

⇒ **flussi di controllo**, che rappresentano il flusso di controllo attraverso l'attività (frecce tra nodi)

⇒ **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

Sviluppare Diagramma di Attività a più livelli di Astrazione (Decomposizione Funzionale)

⇒ chiama un'operazione

Stampa ordine  
(Ordine::stampa)

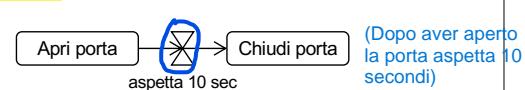
Come Nodo che chiama un Comportamento: indica in più quale classe e quale funzione viene chiamata

### Nodo azione di accettazione evento temporale

(Per Modellare Tempo Relativo, per temporizzare)

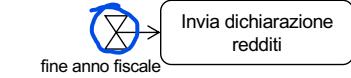
⇒ produce un evento temporale ogni volta che la condizione temporale diventa vera

(Per spaziatura temporale tra due nodi azione)



⇒ diventa attivo solo quando si attiva l'arco

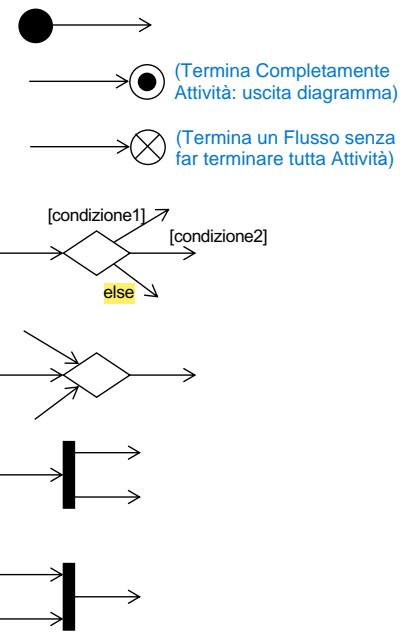
(Quando si scatena evento temporale, avviene passaggio al prossimo nodo)



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 (Utile in caso di Concorrenza e/o Caso di Errore)
- Nodo decisione:** divide il flusso in più flussi alternativi (Esprime IF)
- Nodo fusione:** ricongiunge i flussi a valle di un nodo decisione
- Nodo biforcazione:** divide il flusso in più flussi concorrenti (Rappresenta Concorrenza)
- Nodo ricongiunzione:** sincronizza flussi concorrenti (Sincronizzazione tra Flussi Concorrenti)



112

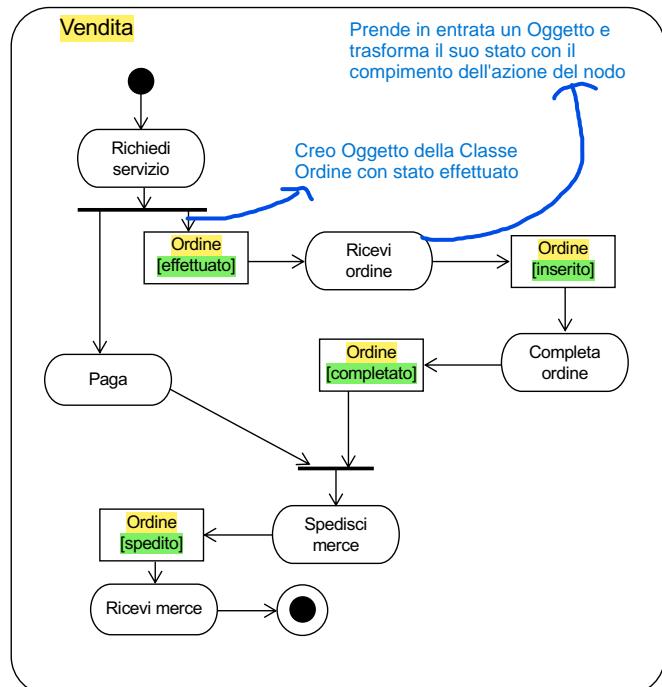
Diagramma di Attività ci permette di mischiare aspetti dinamici e funzionali, ossia di accoppiare alla specifica dei flussi di controllo, anche una specifica funzionale, dove una certa azione consuma dei dati e produce altri dati (dove dati significa oggetti: cioè consuma oggetti e produce oggetti)

## Nodi oggetto

Nodi oggetto li posso usare per collegare due nodi Azione per dire che quelle due azioni si scambiano gli oggetti

- 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 [...] indicano lo stato

Gli Aspetti Funzionali sono dati dai Rettangoli, perché ho dei processi (nodi) che consumano dati (oggetti) e generano dati (oggetti).



Il Diagramma degli Stati di Ordine avrebbe 4 Stati (Effettuato, Inserito, Completato, Spedito) ed avrei i nodi azione come Eventi nel Diagramma degli Stati

113

Servono per mostrare chi fa che cosa

Un modo per partizionare il Diagramma di Attività (per ruolo o unità organizzativa)

Corsie

L'Attività rappresenta il tutt'uno, il contesto, mentre i nodi sono azioni

Nodi Controllo e Nodi Oggetto possono essere posizionate dove ci pare, ma non i Nodi Azione

- 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

dimensione

Ruolo

Vendita		
Cliente	Ufficio vendite	Magazzino
<p>Richiedi servizio</p> <p>Paga</p>	<p>Ricevi ordine</p>	<p>Completa ordine</p>
		<p>Spedisci merce</p> <p>Ricevi merce</p>

corsia

la dimensione di ciascun insieme di corsie (o swimlanes) rappresenta un aspetto del sistema che contribuisce a organizzare e dare significato ai flussi di lavoro o alle attività in corso.

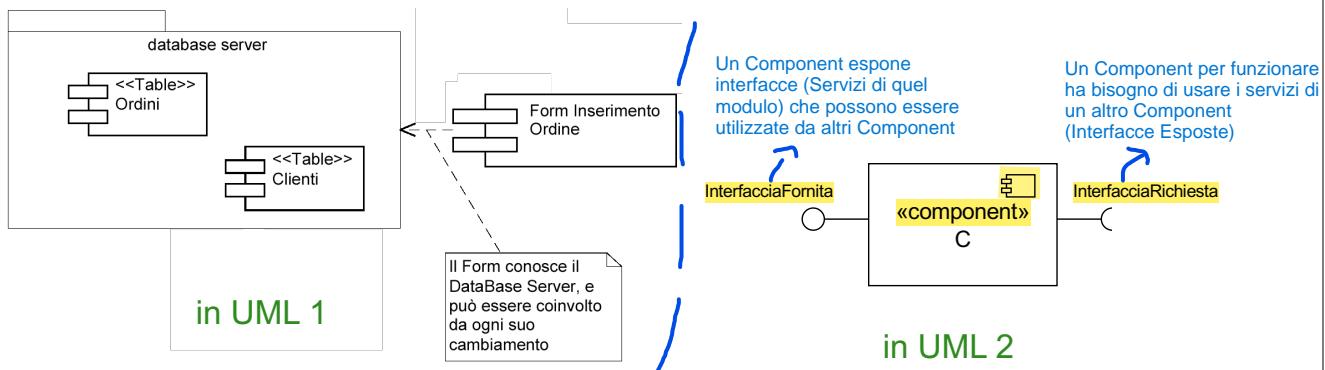
114

# 10

# Diagramma dei componenti

Mostra Architettura Software

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



Interfaccia = Classe Astratta (Non Istanziabile) Stereotipata (Senza Struttura, cioè senza implementazione delle operazioni)

115

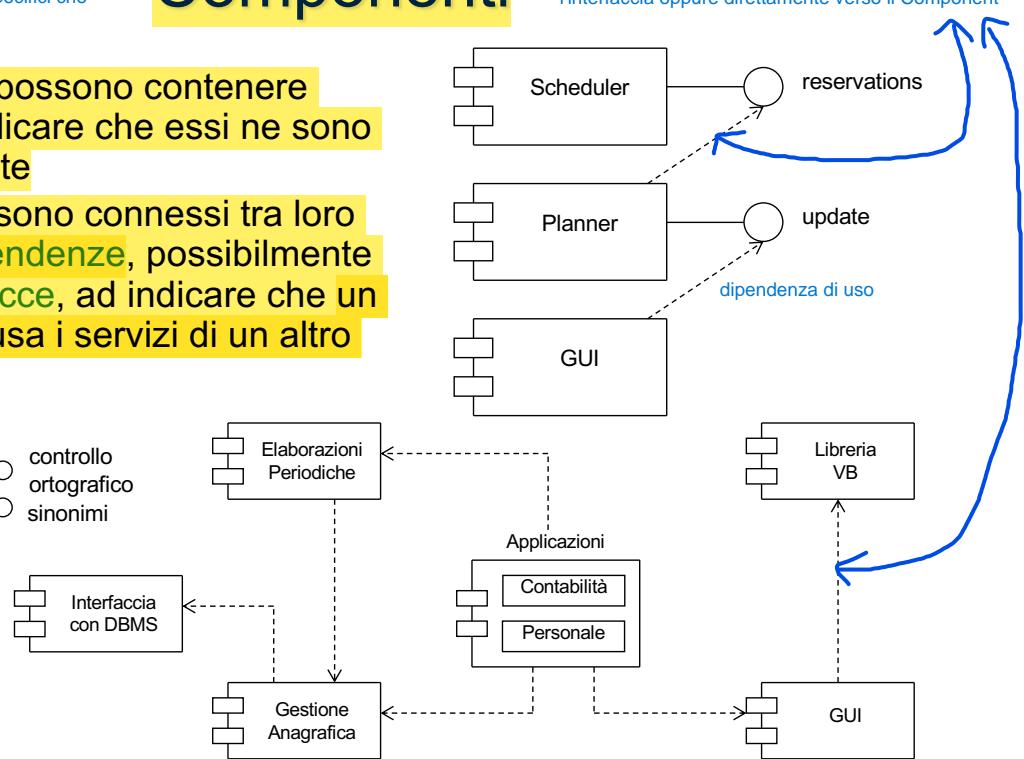
Un componente può essere visto come una "scatola" che incapsula una parte del sistema. Gli oggetti all'interno di un componente rappresentano elementi specifici che fanno parte di quel componente.

## Componenti

La relazione di dipendenza d'uso può essere diretta verso l'interfaccia oppure direttamente verso il Component

- 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

Esempio:  
Un componente chiamato "Gestione Ordini" potrebbe contenere oggetti come Ordine, Cliente e Prodotto. Questi oggetti sono parti del componente e contribuiscono alla sua funzionalità.

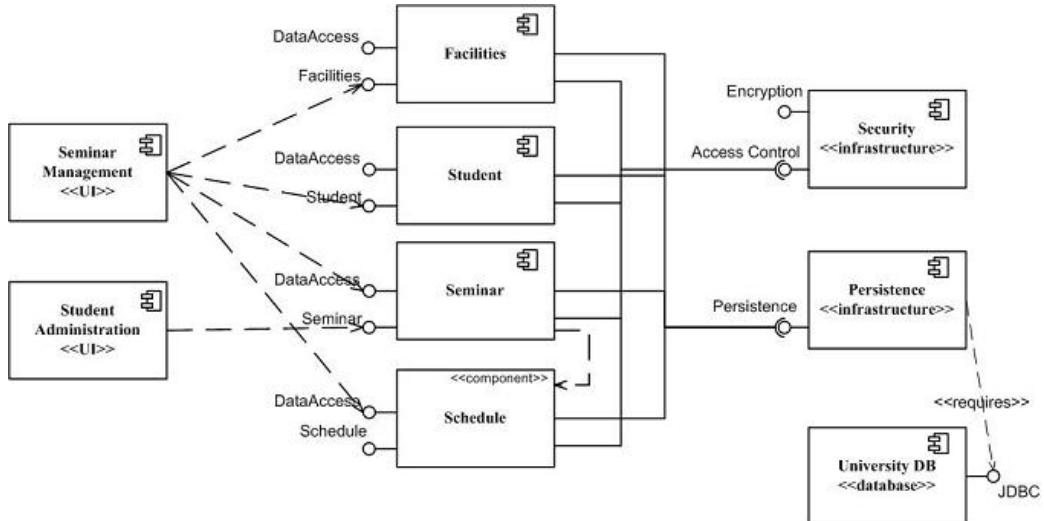


116

Interfacce Esposte = sono gruppi di operazioni esposte da un Component

## Esempio

Misto tra UML 1 e 2



117

(Non ci sono Esercizi su Deployment Diagram, solo quiz)

11

## Diagramma di deployment

Mostra Architettura Hardware

(Può Mostrare anche i Component del Component Diagram)

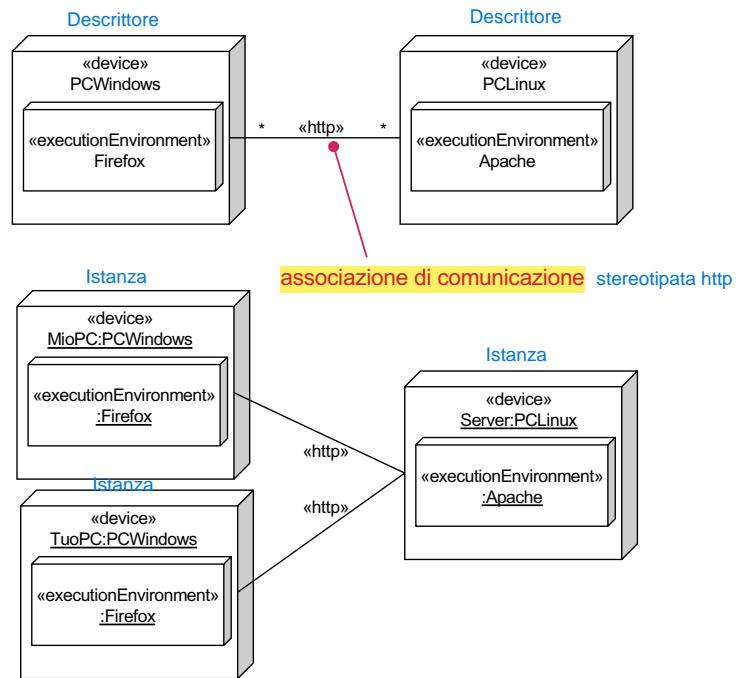
- Specifica l'hardware su cui verrà eseguito il software e il modo in cui il software è dislocato sull'hardware

- 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

(architettura di massima, generale)

(architettura specifica)

Stessa differenza che c'è tra Classe e Oggetto  
(ES: Classe Persona, Oggetto Persona Specifica)



118

## Nodi

- nodi vengono rappresentati con cubi
  - Un **nodo** rappresenta un tipo di **risorsa computazionale** su cui i manufatti possono essere dislocati per l'esecuzione (che hanno un minimo di capacità computazionale)
  - Due stereotipi standard:
    - ⇒ «device» rappresenta un tipo di periferica fisica, per esempio un PC (Oggetto Fisico)
    - ⇒ «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 (Associazioni di Comunicazione)
  - Si possono usare ulteriori stereotipi e icone per aumentare la leggibilità del diagramma
- A differenza di quelli dei Casi d'uso, possono essere stereotipate e possono avere cardinalità agli estremi

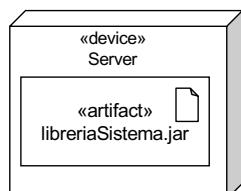
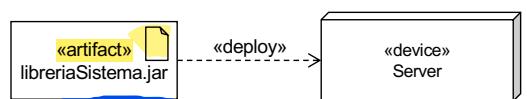
119

(Artifact)

## 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

File di Varia Natura



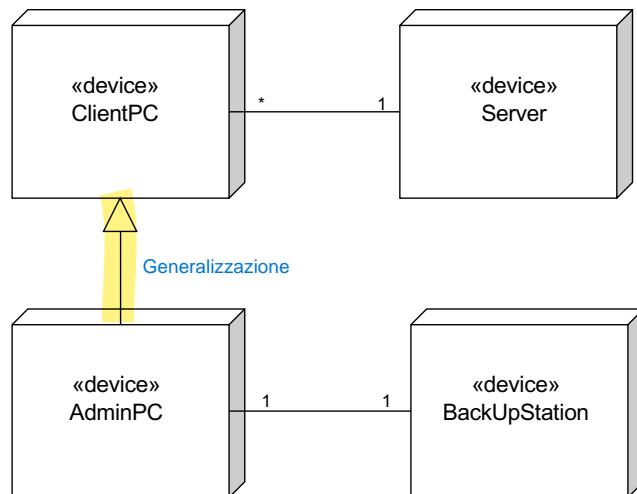
Così mostro in quale Component (Modulo) si manifesta il Manufatto (rispetto alla rappresentazione di Destra, con la parte cerchiata, mostro più dettaglio: senza sarebbe lo stesso di Destra)

Mostro che il Manufatto manifesta (implementa) un Componente del Diagramma dei Componenti (viene mostrato il File Fisico che Implementa quel Component)

(Non è possibile mettere direttamente un Component all'interno di Device )

120

## Esempio



121

## 12

## Benefici di UML

- Superamento della "guerra dei metodi" (UML ha unificato i principali metodi esistenti in un unico linguaggio standard, superando le differenze e le rivalità tra i 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 (Adatto per Modellare Sistemi di Qualsiasi Difficoltà)
  - ↳ 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

UML incoraggia a concentrarsi sui processi e sugli approcci di progettazione

↳ ritorno di attenzione sul processo di lavoro e sugli approcci utilizzati, non solo sulle tecnologie

UML è basato su un metamodello comune, cioè una struttura concettuale condivisa che definisce come gli elementi del linguaggio devono essere rappresentati e interagire tra loro.

↳ 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

(UML è solo un Linguaggio, non un Metodo)

123

## Personalizzazioni

(UML Molto Personalizzabile tramite i Profili)

- 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