

Copyright © 2004 Moreno Marzolla

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 Italy License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.5/it/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

## UML

- Cos'è UML?
  - Il successore di una serie di metodologie di analisi e progettazione orientate agli oggetti
  - UML è un linguaggio (notazione) di modellazione, *non* una metodologia
  - Una metodologia di solito comprende un linguaggio di modellazione e un processo
    - **Linguaggio di modellazione**: notazione (anche grafica) usata per esprimere le caratteristiche di un progetto
    - **Processo**: serie di “consigli” riguardanti i passi da intraprendere per produrre il progetto stesso.

## UML

- UML definisce una **notazione** e un **meta-modello**
- **Notazione**
  - Definisce l'aspetto grafico dei modelli
  - Rappresenta la sintassi del linguaggio di modellazione
  - La notazione adottata è volutamente “informale”
- **Meta-modello**
  - Un insieme di diagrammi che definisce la notazione stessa

## Tipi di diagrammi UML

- Diagrammi Statici
  - Diagrammi di Casi d'Uso
  - Diagrammi di Classe / degli Oggetti / dei Package
  - Diagrammi di Deployment / dei Componenti
- Diagrammi Dinamici
  - Diagrammi di Sequenza
  - Diagrammi di Collaborazione
  - Diagrammi di Attività
  - Diagrammi di Stato

## UML e questo corso...

- La maggior parte delle situazioni può essere modellata con un sottoinsieme ristretto di diagrammi UML.
- Per chi è interessato ad approfondire:
  - Martin Fowler *UML Distilled (Terza Edizione)* Addison-Wesley; ISBN: 8871922077; terza edizione (febbraio 2004)
  - Grady Booch, Ivar Jacobson, James Rumbaugh *The Unified Modeling Language User Guide* Addison-Wesley Pub Co; ISBN: 0201571684; 1st edition (September 30, 1998)
  - Jim Arlow, Ila Neustadt, *UML e Unified Process—Analisi e progettazione Object Oriented*, McGraw-Hill, 2002, ISBN 88-386-6144-8
  - <http://www.omg.org/technology/documents/formal/uml.htm>

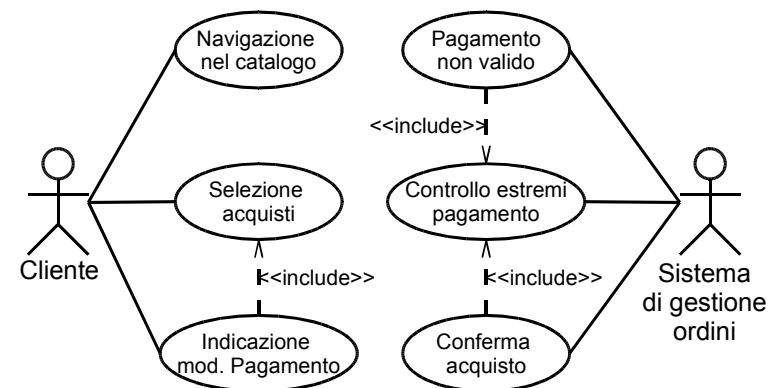
## Diagrammi di Casi d'Uso

- Concetto di **scenario**
  - E' una sequenza di passi che descrivono l'interazione tra un utente e un sistema
  - **Esempio:** Il cliente naviga nel catalogo e raccoglie gli articoli desiderati in un carrello della spesa. Quando il cliente desidera pagare, descrive la modalità di spedizione e fornisce la necessaria informazione riguardante la propria carta di credito prima di confermare l'acquisto. Il sistema controlla se la carta di credito è valida e conferma l'acquisto sia immediatamente che con un successivo messaggio di posta elettronica

## Casi d'Uso

- Un **caso d'uso** è un insieme di azioni legate da un obiettivo comune per l'utente
- Esempio: per modellare l'acquisto di un prodotto si possono considerare diversi casi d'uso
  - Conferma Acquisto
  - Pagamento non valido
- I diagrammi di caso d'uso descrivono ad alto livello l'interazione tra il sistema e uno o più *attori* che richiedono un servizio

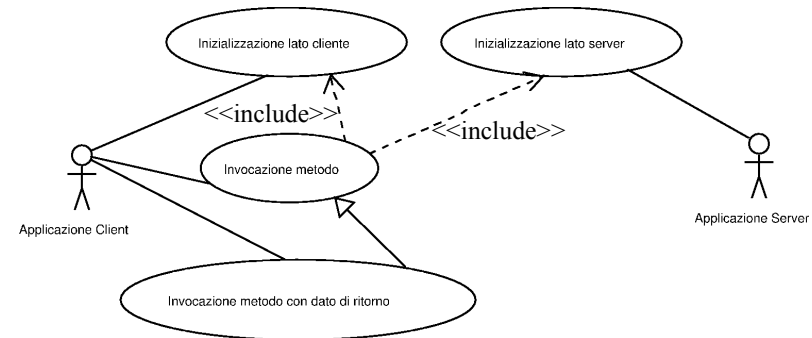
## Esempio: Acquisto di un prodotto



## Esempio: CORBA

- Esistono un server e un client che devono comunicare tramite CORBA
- Il client e il server devono eseguire una inizializzazione, quando partono
- Un client può fare due cose:
  - Richiamare un metodo remoto, senza restituzione del risultato
  - Richiamare un metodo remoto, con restituzione di un risultato
  - In entrambi i casi, queste operazioni possono essere svolte solo se il client e il server sono stati inizializzati

## Esempio: CORBA

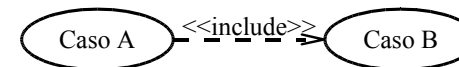


## Casi d'uso: Attori

- Un **attore** è un ruolo interpretato dall'utente nei confronti del sistema.
- Gli attori non devono necessariamente essere persone
- Gli attori interpretano casi d'uso
  - Un singolo attore può eseguire più casi d'uso
  - Un caso d'uso potrebbe essere svolto da più attori

## Relazioni tra casi d'uso

- Inclusione
  - Un determinato comportamento si ripete in più casi d'uso

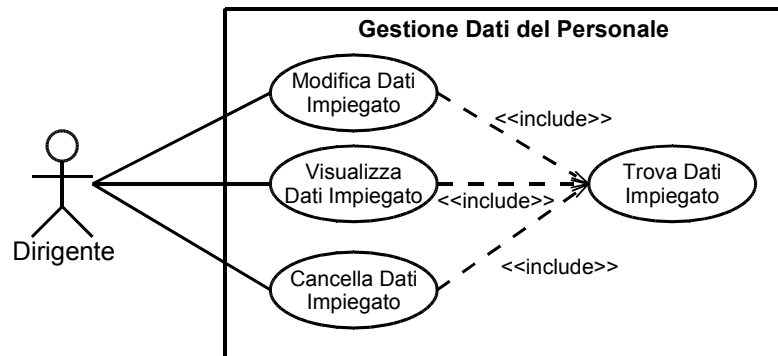


- Generalizzazione

- Un caso d'uso è simile ad un altro, ma fa qualcosa di più



## Esempio <<include>>



## Individuare gli attori / 1

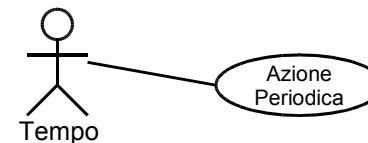
- Gli attori sono sempre esterni al sistema
  - Quindi, non possono essere controllati
- Gli attori interagiscono direttamente con il sistema
  - In tal modo aiutano a fissare i confini del sistema
- Gli attori rappresentano i ruoli generici che persone o cose possono rivestire nei confronti del sistema
  - NON rappresentano persone o cose specifiche

## Individuare gli attori / 2

- Una stessa entità può rivestire (contemporaneamente o in tempi diversi) più ruoli nei confronti del sistema
- Ogni attore deve essere identificato con un nome breve e significativo
- Ogni attore deve essere caratterizzato da una descrizione significativa

## Il tempo come attore

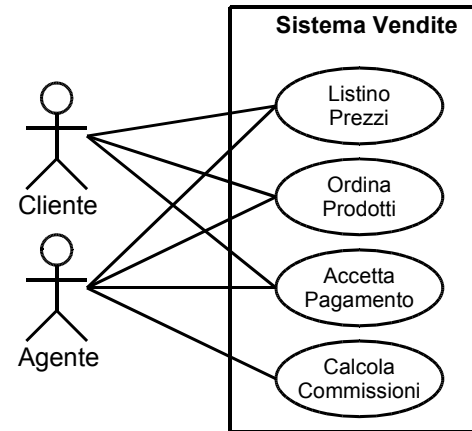
- E' possibile modellare azioni periodiche, che apparentemente non sono attivate da nessuno, inserendo un attore fittizio
  - Es, azioni periodiche come backup o simili



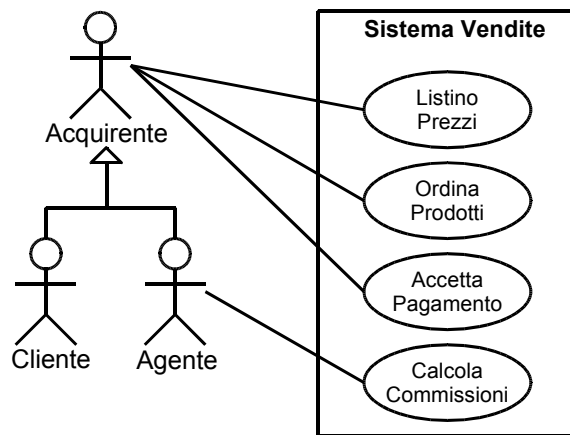
## Modellazione avanzata

- E' possibile usare il concetto di *ereditarietà* anche con i diagrammi di casi d'uso
- Lo scopo è di rendere più comprensibile la descrizione degli attori, e identificare le relazioni tra attori diversi

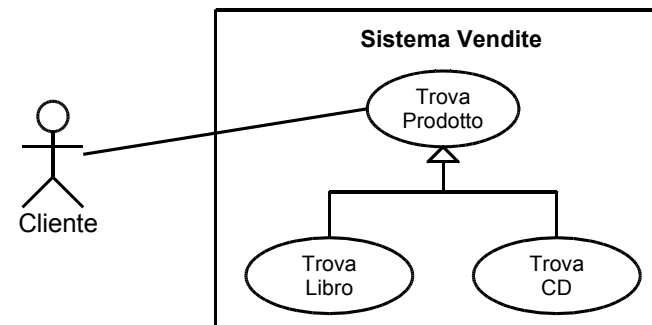
## Esempio / 1



## Esempio / 2



## Generalizzazione tra casi d'uso





## Molteplicità delle Associazioni

- Gli estremi di una associazione possono essere etichettati con il **nome del ruolo**
  - Se non è indicato, il ruolo è il nome della classe
- Il capo di una associazione ha una **molteplicità**
  - Indica quanti oggetti possono prendere parte alla relazione
  - La molteplicità si può anche indicare con un intervallo m..n
  - L'asterisco \* rappresenta l'intervallo da 0 a infinito
  - Nella pratica, le molteplicità più usate sono 1, \* e 0..1

## Navigabilità delle Associazioni

- Le associazioni possono avere una **navigabilità**
  - Quale degli oggetti agli estremi dell'associazione è responsabile per segnalare la sua relazione con l'altro
  - Esempio: gli oggetti della classe Ordine sono responsabili per segnalare a quali oggetti Cliente appartengono

## Attributi e Operazioni

- Concetto intuitivo: l'attributo "nome" della classe Cliente indica che i clienti hanno un nome
  - Gli attributi individuano lo *stato* interno degli oggetti di una data classe
- Le **operazioni** rappresentano le computazioni che una classe sa come effettuare
  - Possono riportare all'esterno informazioni sullo stato degli oggetti
  - Possono modificare lo stato degli oggetti

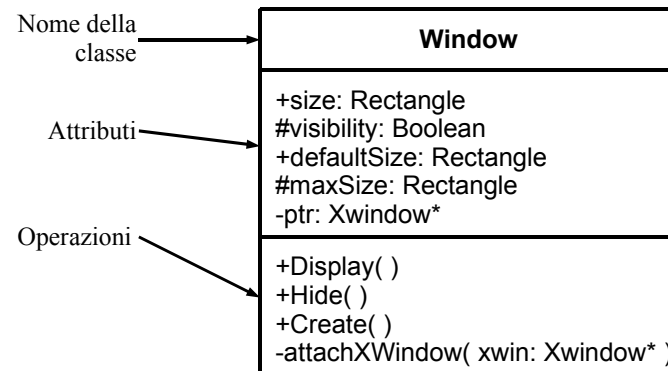
## Sintassi delle operazioni

- *visibilità nome {lista-parametri}: tipi-ritornati {stringa-proprietà}*
- **Visibilità:**
  - + (pubblica)
  - # (protetta)
  - - (privata)
- **Nome**
  - Una stringa che indica il nome dell'operazione

## Sintassi delle operazioni (2)

- Lista-Parametri
  - Contiene una serie di parametri separati da virgole
  - *direzione nome: tipo=valore*
  - Direzione può essere input (*in*), output (*out*), o entrambi (*inout*). Default *in*
- Tipi-ritornati
  - Lista di tipi di ritorno separata da virgole (normalmente si tratterà di un singolo tipo)
- Stringa-proprietà
  - Indica i valori delle proprietà che si applicano all'operazione data

## Esempio

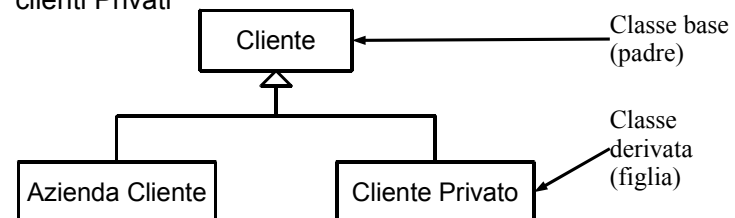


## Visibilità

- Gli elementi di una classe (attributi e operatori) possono essere pubblici, privati o protetti
- E' indispensabile capire come questo concetto viene implementato nei linguaggi di programmazione, ad es. Il C++
  - Un membro *pubblico* (+) è visibile ovunque nell'intero programma e può essere chiamato da qualunque oggetto del sistema
  - Un membro *protetto* (#) può essere usato dalla classe che lo definisce o da una sua sottoclasse
  - Un membro *privato* (-) può essere usato solo all'interno della classe che lo definisce

## Generalizzazione (sottotipo)

- Indica che una classe è sottoclasse di un'altra. Le sottoclassi individuano entità più specializzate rispetto alle classi base
  - Esempio: I clienti possono essere clienti Aziendali o clienti Privati



## Aggregazione e Composizione

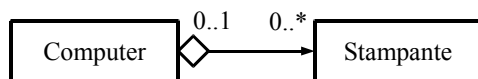
- L'**aggregazione** è la relazione parte-di
  - E' come dire che una macchina ha un motore e quattro ruote come sue parti
- La **composizione** è simile all'aggregazione, ma
  - Ci si aspetta che l'oggetto "parte" appartenga ad un solo oggetto "tutto"
  - Ci si aspetta che le parti abbiano lo stesso ciclo di vita del tutto. La cancellazione dell'oggetto composto si propaga a tutti i suoi oggetti parte

## Aggregazione

- L'aggregato può in alcuni casi esistere indipendentemente dalle parti, in altri casi no
- Le parti possono esistere indipendentemente dall'aggregato
- L'aggregato è in qualche modo incompleto se mancano alcune delle sue parti
- E' possibile che più aggregati condividano una stessa parte

## Esempio

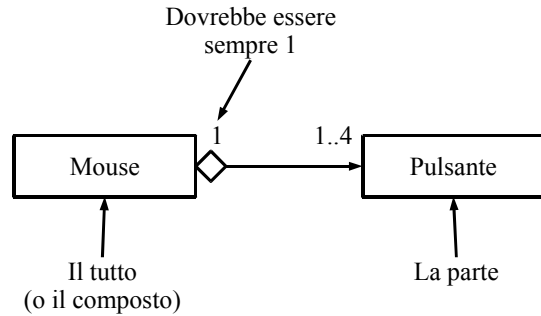
- Un **computer** può essere collegato a 0 o più **stampanti**
- In un qualunque istante una **stampante** è collegata a 0 o 1 **computer**
- Nel corso del tempo, molti **computer** possono utilizzare la stessa **stampante**
- La **stampante** può esistere anche se non è collegata ad alcun **computer**
- La **stampante** è indipendente dal **computer**



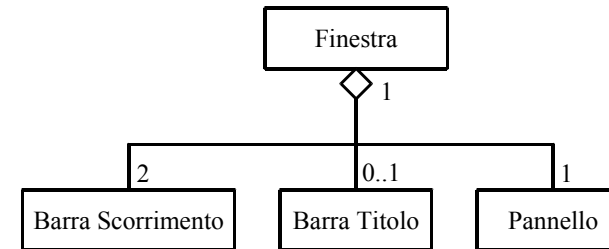
## Composizione

- Ogni parte può appartenere a un solo composito per volta
- Il composito è l'unico responsabile di tutte le sue parti
  - E' responsabile della loro creazione o distruzione
- Il composito può anche rilasciare una sua parte, a patto che un altro oggetto si prenda la relativa responsabilità
- Se il composito viene distrutto, deve distruggere tutte le sue parti o cederne la responsabilità

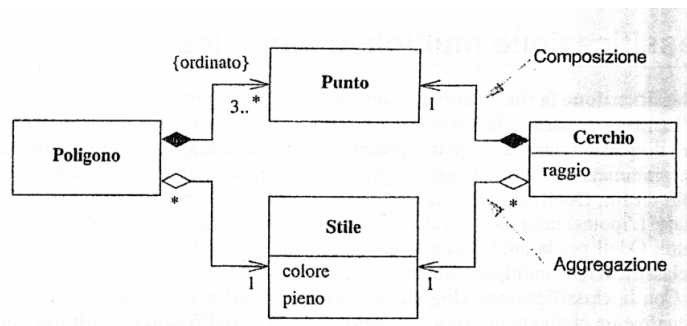
## Esempio / 1



## Esempio / 2



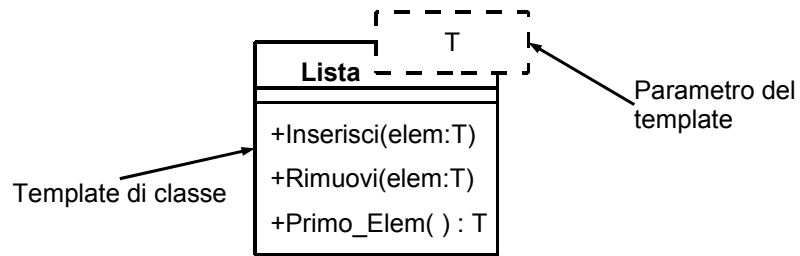
## Aggregazione e Composizione



## Classi parametriche

- Nozione supportata da molti linguaggi (in particolare C++), col nome di **template**
- E' utile per lavorare con "collezioni" o "insiemi di oggetti" in linguaggi fortemente tipizzati
- Ad esempio, consente di definire una "lista di elementi di tipo T" ove T è un parametro della lista

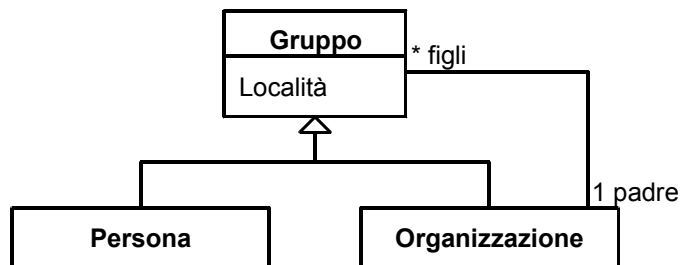
## Classi parametriche in UML



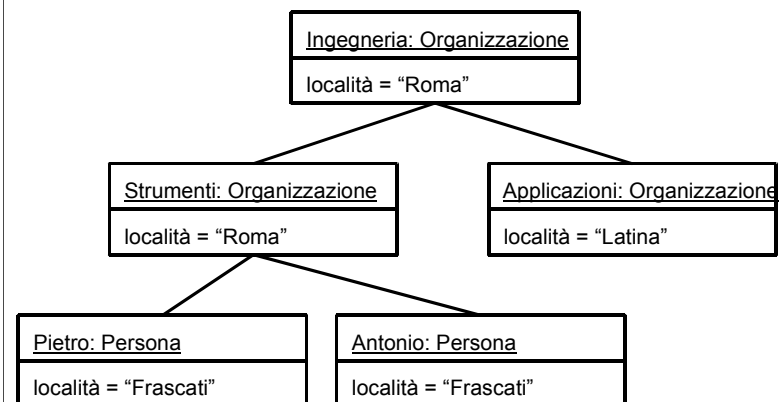
## Diagramma degli oggetti

- Rappresenta l'insieme degli oggetti che compongono un sistema software in un dato momento
- Mostra le *istanze* delle classi, piuttosto che le classi stesse
  - Viene anche chiamato *Diagramma delle istanze*

## Esempio: composizione di un gruppo (Diagramma delle Classi)



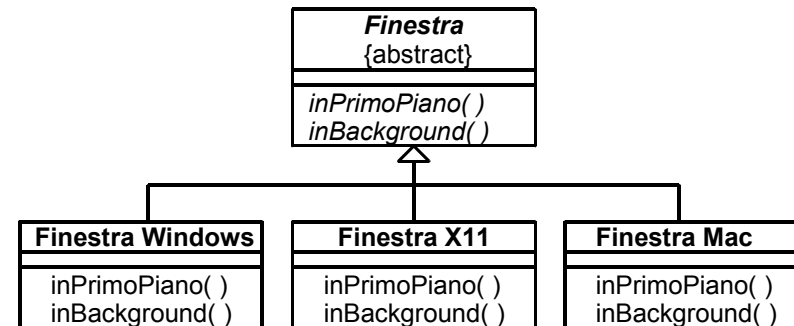
## Esempio: composizione di un gruppo (Diagramma degli Oggetti)



## Interfacce e classi Astratte

- Una classe **astratta** viene utilizzata per definire l'*interfaccia* di un oggetto, senza però specificarne il comportamento
  - Si indicano quali sono le operazioni supportate da oggetti di quella classe, ma non come queste operazioni sono implementate
  - Non è possibile istanziare oggetti della classe astratta
  - La classe viene utilizzata per derivare delle altre classi concrete, che implementano l'interfaccia in modi diversi

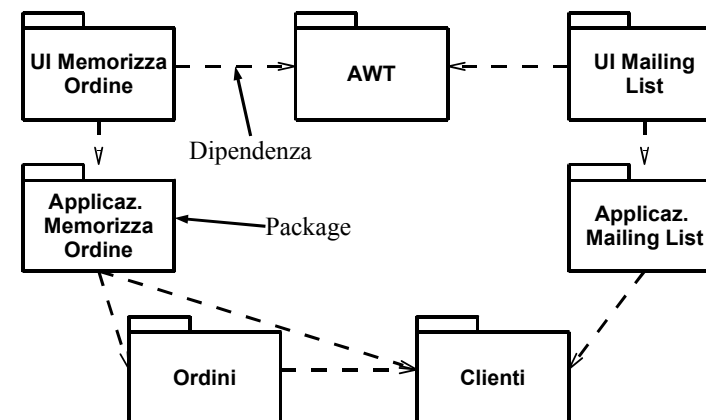
## Interfacce e classi Astratte



## Diagrammi di Package

- Un *package* è un gruppo di elementi del modello
- Un package può contenere altri package e/o altri tipi di elementi
  - Ogni genere di elementi UML può essere organizzata in packages, anche se noi vedremo principalmente i package di classi
- Un *diagramma dei package* mostra i package con le loro dipendenze.
  - Si dice che tra due pacchetti esiste una dipendenza se i cambiamenti apportati all'uno si possono potenzialmente ripercuotere sull'altro

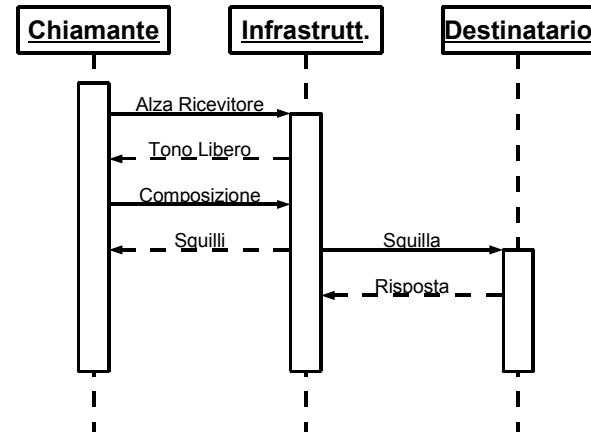
## Esempio



## Diagrammi di Sequenza

- Fino a qui abbiamo visto i diagrammi che rappresentano la struttura *statica* di una Architettura Software
- Occorre però anche caratterizzarne il comportamento *dinamico*
- I Diagrammi di Sequenza sono utilizzati per mostrare quali comunicazioni vengono effettuate tra oggetti.

## Esempio



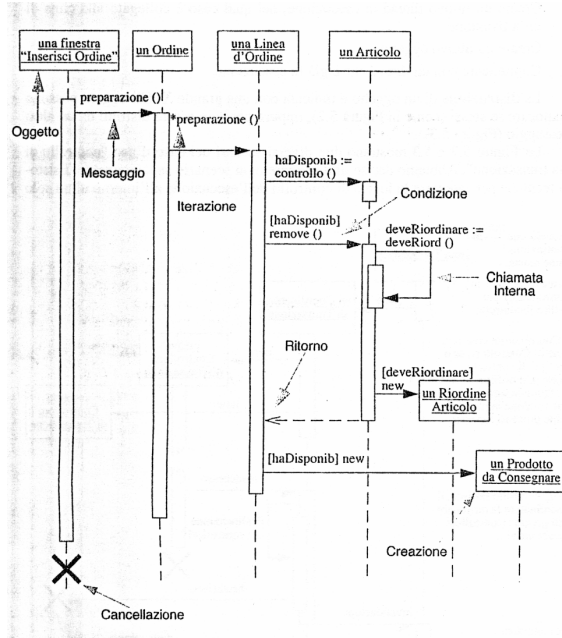
## Diagrammi di Sequenza

- La linea tratteggiata verticale è la *linea di vita* di ciascun oggetto
- Un messaggio è indicato da una freccia tra due linee di vita. Ogni messaggio è etichettato col proprio nome
- Per mostrare quando un oggetto è attivo, viene disegnato un box di attivazione

## Diagrammi di Sequenza (2)

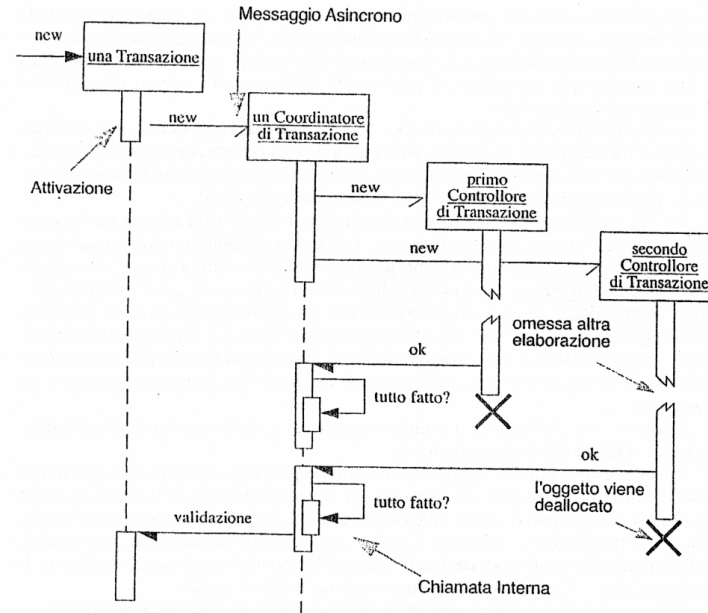
- Si può anche mostrare una chiamata **interna** di un oggetto ad una sua funzione
- Un messaggio può avere una **condizione** (es. [deveRiordinare]) che se vera causa l'invio del messaggio.
- Un messaggio può avere un **indicatore di iterazione** che mostra che un messaggio viene inviato più volte a diversi oggetti (es. \*[per tutte le linee d'ordine])

# Esempio



Moreno Marzolla

# Es.



Moreno Iv

# Diagrammi di Collaborazione

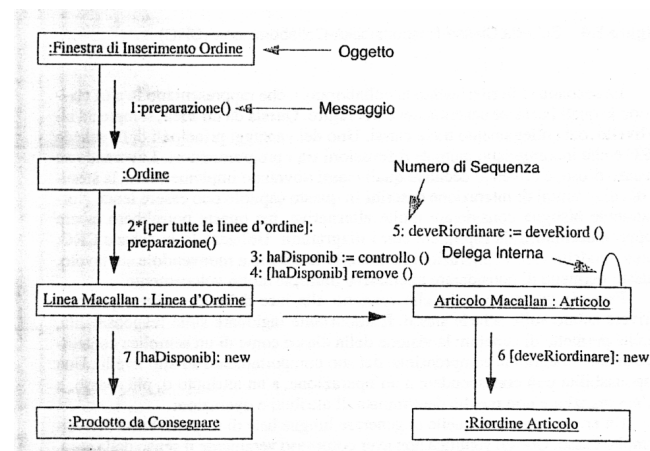
- Gli oggetti sono disegnati come icone
- Le frecce indicano messaggi inviati tra oggetti all'interno di un determinato caso d'uso
- La sequenza dei messaggi è individuata dalla numerazione dei messaggi

Moreno Marzolla

Ingegneria del Software

55

# Esempio



Moreno Marzolla

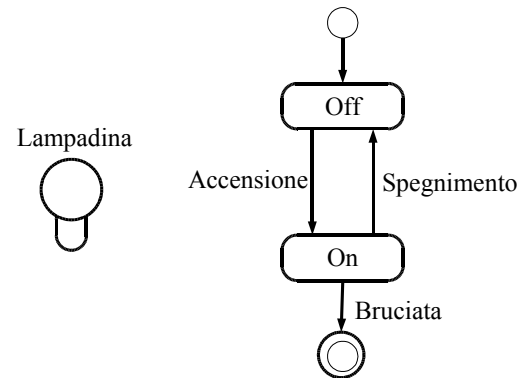
Ingegneria del Software

56

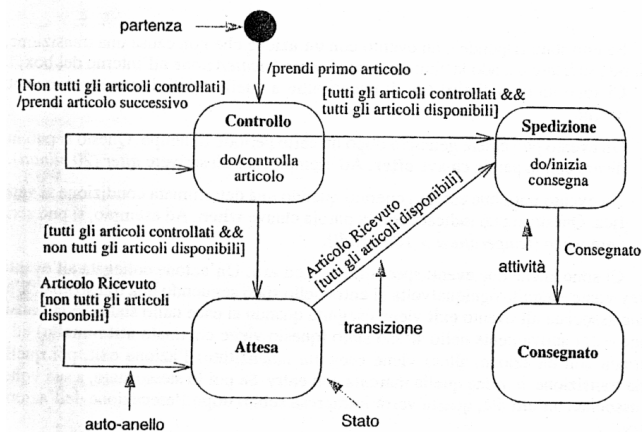
## Diagrammi di Stato

- Descrivono l'evoluzione del sistema
- Sono costituiti da una serie di **stati** che descrivono delle **attività**, tra i quali ci si muove attraverso delle **azioni**
  - Le azioni sono associate a cambiamenti di stato (transizioni), quindi sono considerate processi rapidi e non interrompibili
  - Le attività sono associate a stati, possono prendere un lasso di tempo più lungo e possono essere interrotte da un evento

## Esempio semplice: la lampadina



## Esempio



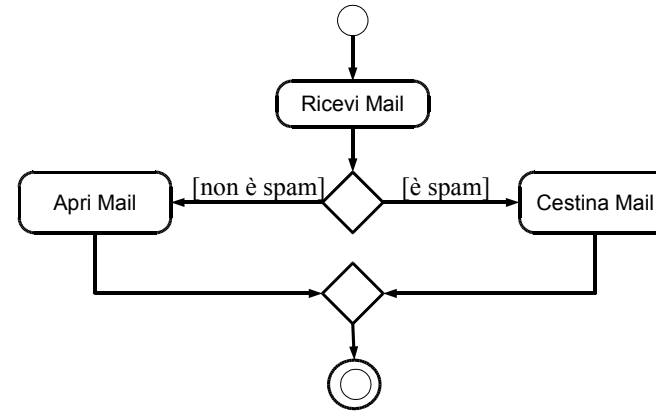
## Transizioni e attività

- La sintassi generale di una etichetta associata ad una transizione è:  
 $\langle \text{Evento} \rangle [\langle \text{Condizione} \rangle] / \langle \text{Azione} \rangle$ 
  - Esegui l'azione se si verifica l'evento, e la condizione è vera
  - Tutte le parti dell'etichetta di una transizione sono opzionali
- La sintassi di una attività è più semplice:  
 $\text{do/} \langle \text{nome attività} \rangle$

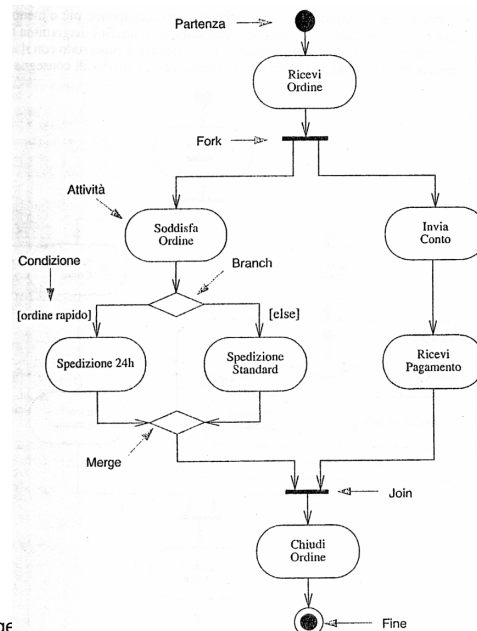
## Diagrammi di attività

- Descrivono la sequenza delle attività e supportano un comportamento sia *condizionale* che *parallelo*
- Due costrutti fondamentali:
  - **Branch** (diramazione)
  - **Merge** (giunzione)

## Decisioni



## Esempio



## Diagrammi di Deployment

- Mostrano le relazioni fisiche tra i componenti software e hardware del sistema finito
- I **nodi** rappresentano un qualche tipo di unità computazionale
  - Un pc, un sensore, un mainframe...
- Le **connessioni** fra i nodi mostrano i canali di comunicazione usati dai componenti del sistema per interagire

## Diagrammi dei Componenti

- Illustrano le varie componenti di un sistema e le loro dipendenze
- Una **componente** normalmente è un modulo software. Spesso coincide con un package
  - Non sempre, dal momento che le componenti rappresentano genericamente un'unità di codice.
  - Di conseguenza, una singola classe potrebbe essere presente in più componenti, ma definita in un solo package

## Diagrammi dei Componenti

- Le **dipendenze** mostrano come i cambiamenti apportati ad uno di essi si ripercuotono sugli altri
  - Esiste una varietà di dipendenze, tra cui
    - Compilazione
    - Comunicazione
  - Spesso le dipendenze mostrano in che modo le componenti comunicano tra loro

## Diagrammi combinati

- Benché i diagrammi dei componenti e di deployment possano essere separati, si possono anche integrare
- E' possibile mostrare quali componenti si trovano in esecuzione all'interno di ogni nodo del diagramma di deployment

## Esempio

