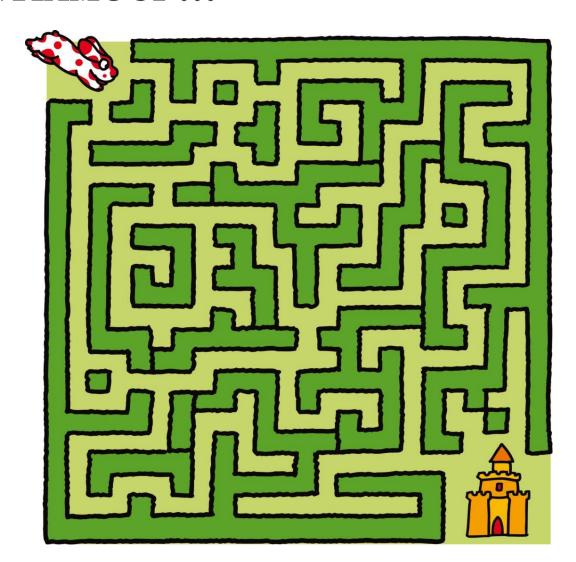


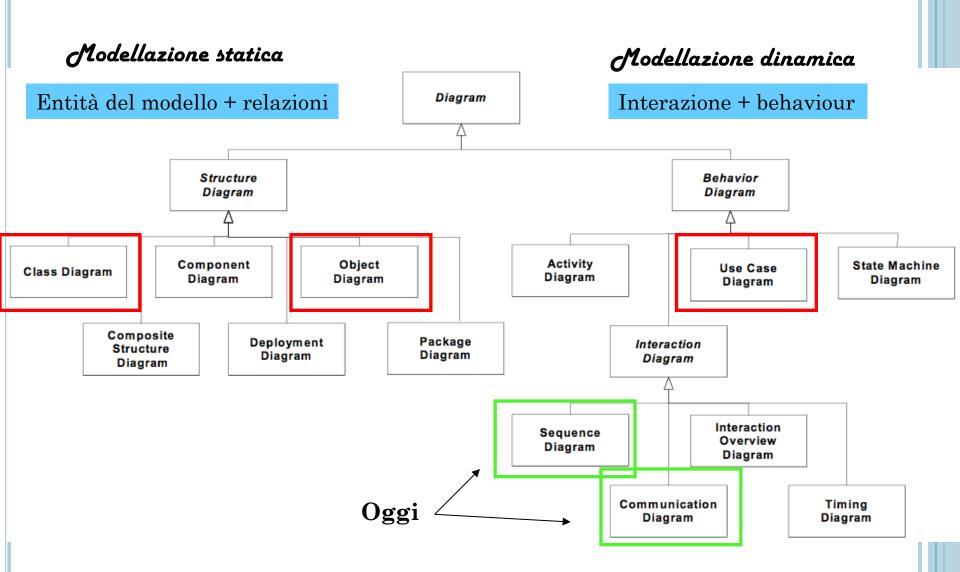
# UML 2.0: SEQUENCE DIAGRAM (+ COMMUNICATION DIAGRAM)

Ingegneria del Software 2024-2025

# ORIENTIAMOCI ...



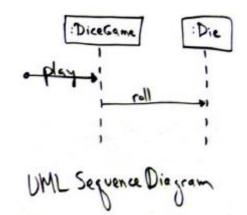
#### ORIENTIAMOCI ...



#### AGENDA

- Diagrammi di interazione
  - Diagrammi di sequenza
    - Cosa descrivono?
    - La notazione
      - I Frame ...
    - Quando e come usarli?
  - Diagrammi di comunicazione
    - Solo cenni

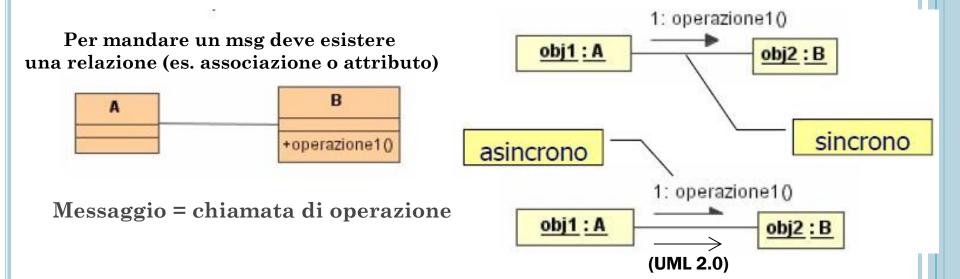
- Esercizio 'Ordine di acquisto'
  - Registrazione AW



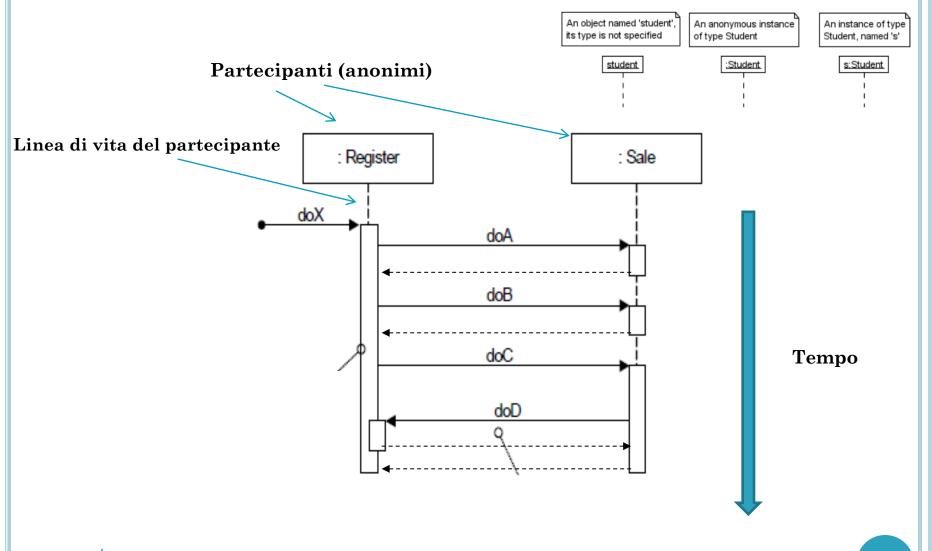
Azienda	ROAR spa				
Indirizzo	Via Grimaldello				
CAP	18017	Città	San Lorenzo al Mare	Prov. IM	
Referente	Filippo Ricca		fono 338-3143**	338-3143***	
Data dell'o	rdine 18/10/201	1			
	Prodotto		Prezzo	Quantità	Importo
Prodotto A			€ 12,00	2	€ 24,0
Prodotto B			€ 10,00	1	€ 10,0
Prodotto C			€ 18,00	3	€ 54,0
Prodotto D			€ 14,00	2	€ 28,0
Prodotto E			€ 6,00	1	€ 6,0
				Totale IMPORTO	€ 122,0
				IVA 20%	€ 24,4
				TOTALE	

#### DIAGRAMMI DI INTERAZIONE

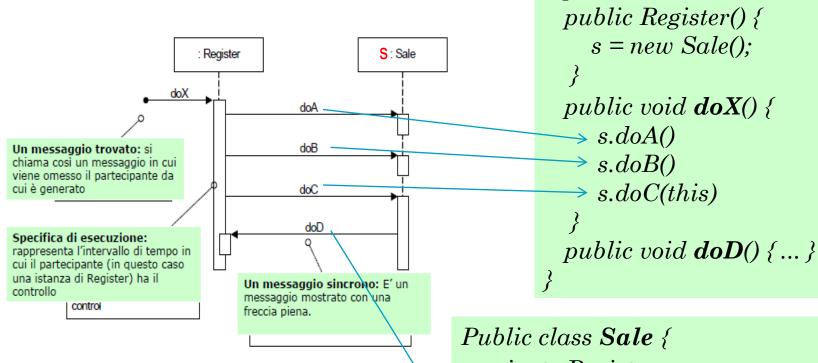
- o I diagrammi di interazione descrivono la collaborazione di un gruppo di oggetti
  - o ovvero lo scambio di messaggi ...
- o I messaggi possono essere
  - **sincroni**: il mittente si pone in attesa del risultato
  - asincroni il mittente continua l'esecuzione
    - o usati per rappresentare i processi concorrenti



# NOTAZIONE SEQUENCE DIAGRAM



#### Possibile implementazione



Nel SD abbiamo omesso: I ritorni e i parametri dei messaggi

```
Public class Sale {
  private Register r;
  public void doA() { ... }
  public void doB() { ... }
  public void doC(Register reg) {
    r=reg;
    r.doD()
  }
```

Public class Register {

private Sale s;

## SINTASSI DEI MESSAGGI

#### returnValue = message(parameters):returnType

parameters è una lista di 'parameter:parameterType' separati da virgola

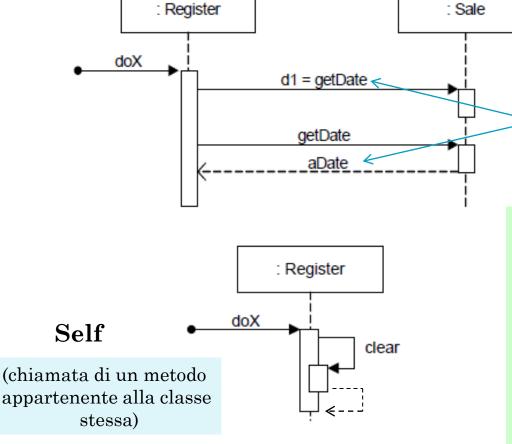
Tutti ammessi!

#### Esempio:

- getProduct
- $\circ$  d = getProduct
- d = getProduct(id)
- d = getProduct(id:ItemID)
- d = getProduct(id:ItemID) : Product



## VALORI DI RITORNO E MESSAGGI SELF

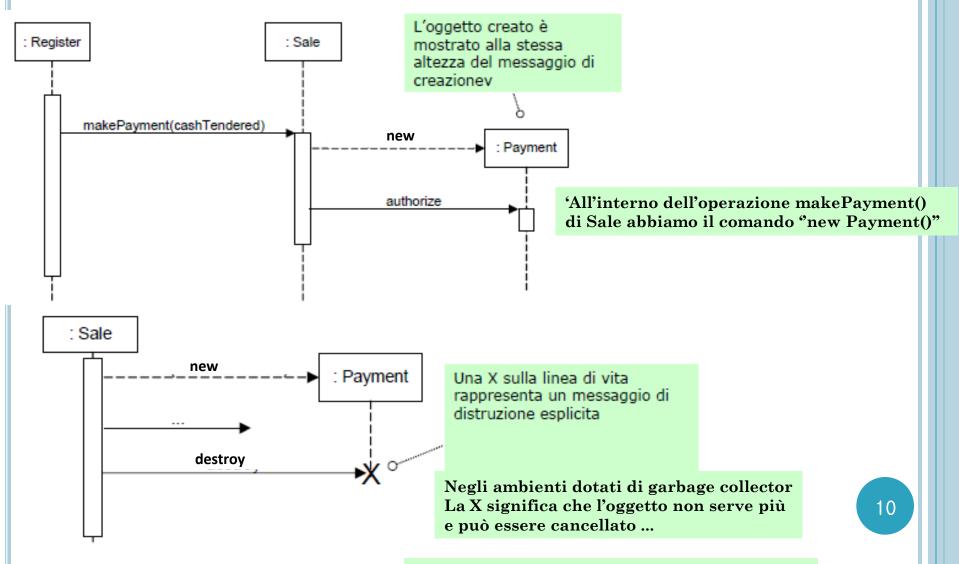


```
Public class Register {
  public void doX() {
     this.clear()
     ...
  }
  public void clear() {
     ....
  }
}
```

Valori di ritorno

due modi equivalenti!!!

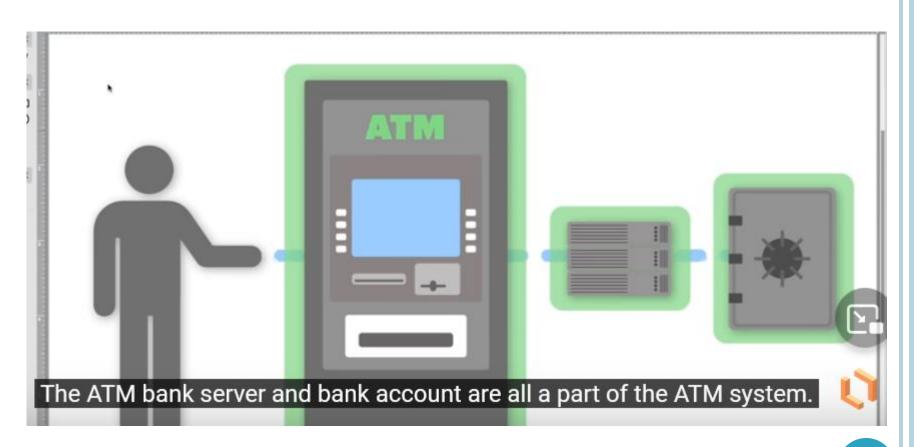
## CREAZIONE E DISTRUZIONE



Un'oggetto si può anche autodistruggere

Come avviene la comunicazione tra i vari partecipanti durante un prelievo di soldi?

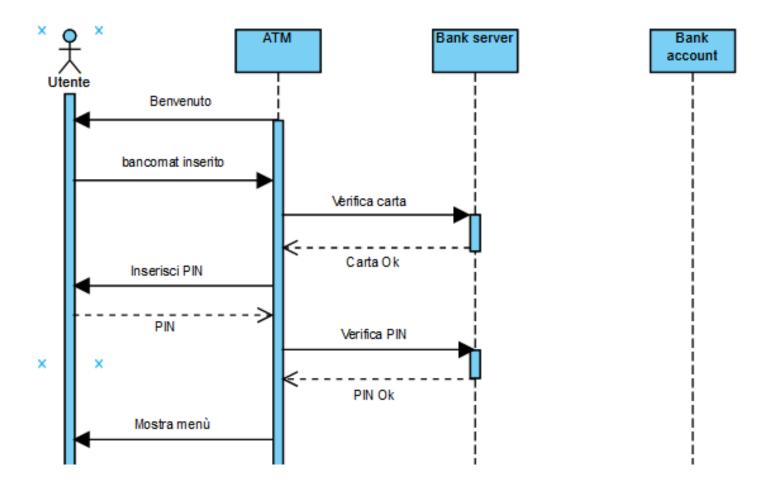
### ESEMPIO: ATM

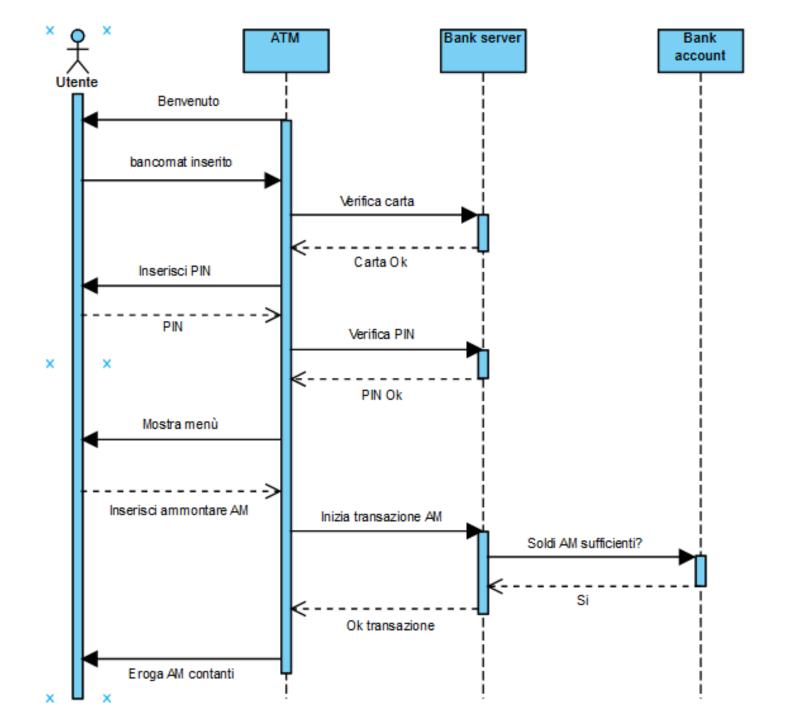


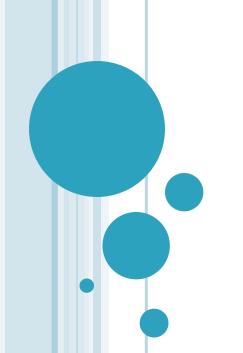
Utente (è un partecipante)

ATM Bank server Bank account

Sistema SW







# UML 2.0: Sequence Diagram: Frame

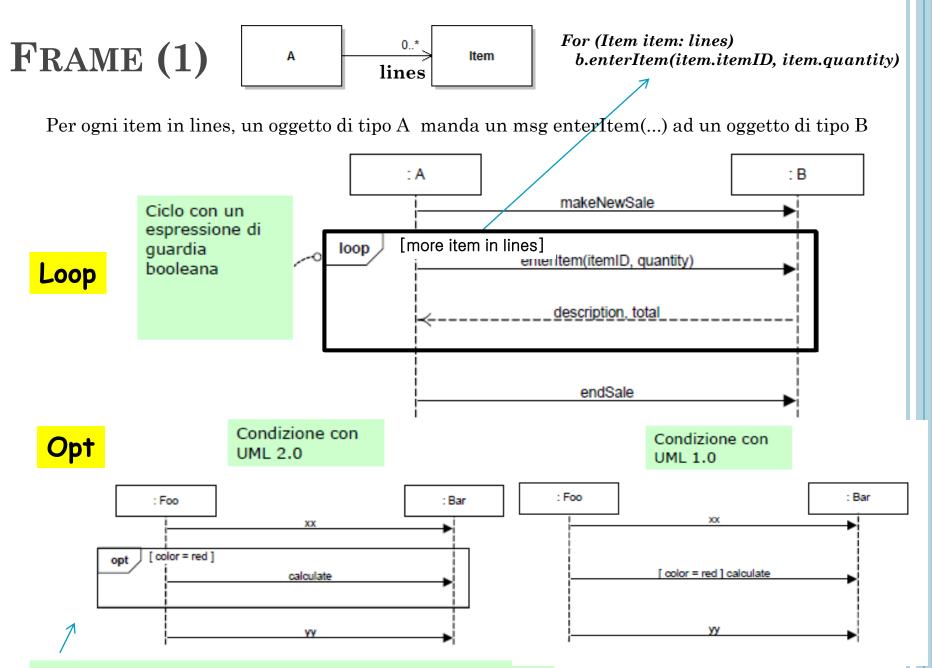
### CICLI CONDIZIONI E ALTRO ....

- Cicli e condizioni (la logica di controllo) possono essere espressi mediante i frame di interazione
  - Una sorta di "cornice" che serve ad inquadrare una parte del diagramma

Copzionale (dipende dal tipo di operatore)

operatore [guardia]

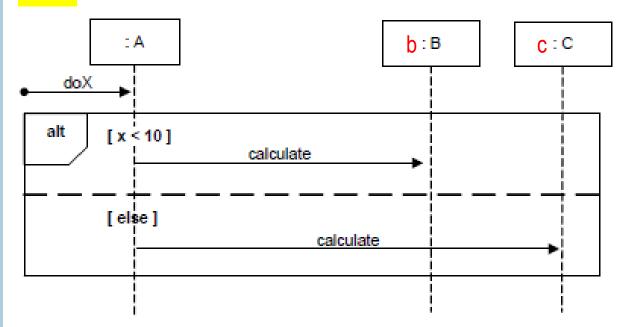
< Diagram's Content Area >



Il frame viene eseguito solo se la condizione è valida

# FRAME (2): IF THEN ELSE / CASE

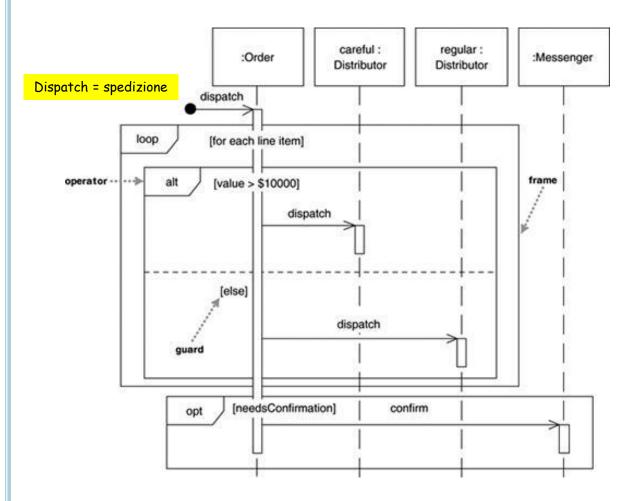
#### Alt



Si possono aggiungere più alternative (corrisponde al Case)

```
Public class A {
 private B b;
 private C c;
 private int x;
 public A() \{
    b = new B();
    c = new C();
    x = \dots
 public void doX() {
    if x < 10 {
      b.calculate()
     else{}
      c.calculate()
```

# FRAME (3): FRAME ANNIDATI



#### ORDINE DI ACQUISTO Azienda ROAR spa Via Grimaldello Città San Lorenzo al Mare Telefono 338-3143\*\*\* Referente Filippo Ricca Data dell'ordine 18/10/2011 Prodotto A € 12,00 € 24,00 Prodotto B € 10,00 € 10,00 € 54,00 Prodotto C € 18,00 Prodotto D € 14,00 € 28,00 € 6,00

Totale IMPORTO

TOTALE

Ordine				
numero: int				
data Data				
spedizione()				
	1*			
Linea d'ordine				
quantità: int				
valore: Dollari				

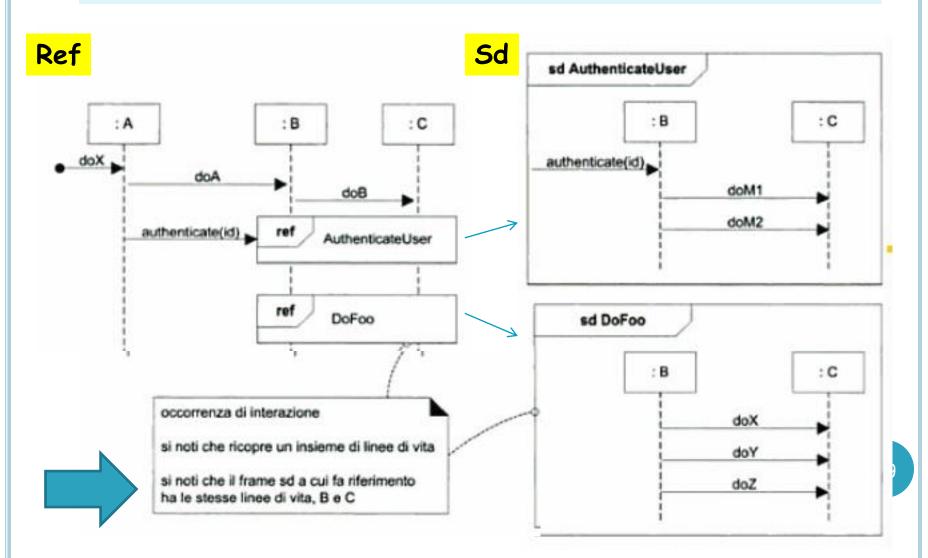
€ 122,00

€ 24,40

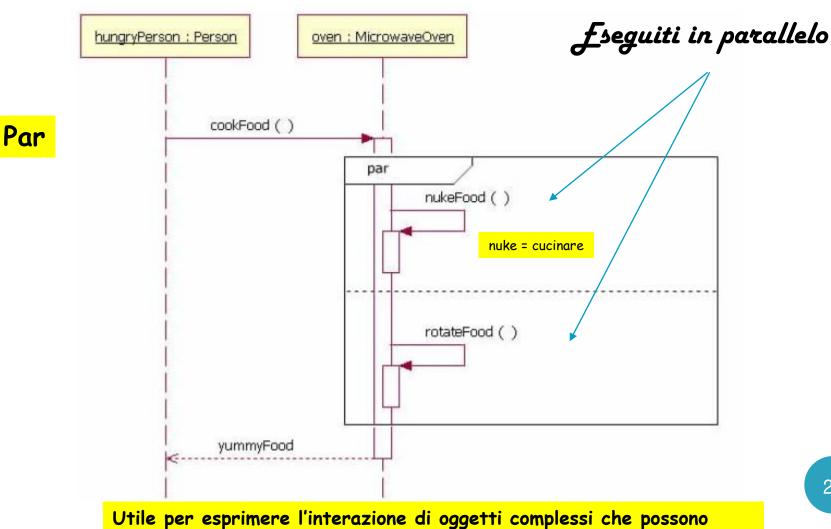
€ 146,40

# FRAME (4): REF

Ref (sta per reference) è usato per semplificare i sequence diagram. Simile al concetto di "procedura" in un linguaggio di programmazione



# FRAME (5): PAR



eseguire diverse azioni/operazioni in parallelo (es. Forno a microonde)

20

#### **OPERATORI USATI CON I FRAME**

sd

alt Alternative multiple fragments; only the one whose condition is true will execute (<u>Figure</u> 4.4). **If** - then - else opt Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace (Figure 4.4). Parallel; each fragment is run in parallel. par loop Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration (Figure 4.4). region Critical region; the fragment can have only one thread executing it at once. ref Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.

Sequence diagram; used to surround an entire sequence diagram, if you wish.





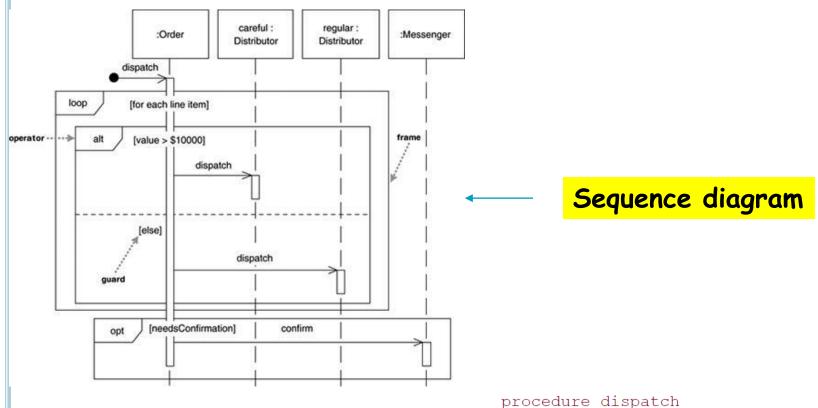
#### PRO E CONTRO

- I diagrammi di sequenza:
  - Sono un ottimo mezzo per visualizzare l'interazione tra oggetti, non la logica di controllo
    - non buoni per spiegare i dettagli degli algoritmi, come i cicli e i comportamenti condizionali
      - Volendo cicli e condizioni sono aggiunti mediante i frame
    - o ma rendono chiare le chiamate tra partecipanti ed illustrano bene cosa fa ognuno di essi ...

#### • M. Fowler suggerisce:

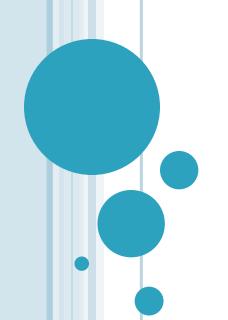
- o di limitare l'uso dei frame di interazione
  - o sono molto pesanti
- per la logica di controllo usare lo **pseudocodice**!
  - oppure utilizzare + diagrammi di sequenza
    - uno per ogni alternativa (scenario)

# PSEUDOCODICE VS. SEQUENCE DIAGRAM



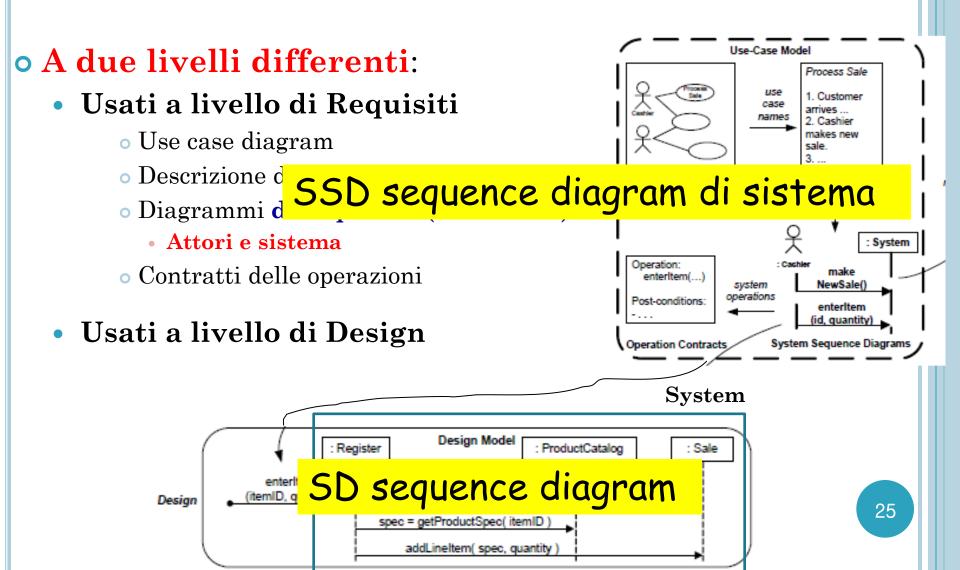
Pseudocodice

```
foredure dispatch
  foreach (lineitem)
    if (product.value > $10K)
        careful.dispatch
    else
        regular.dispatch
    end if
    end for
    if (needsConfirmation) messenger.confirm
end procedure
```

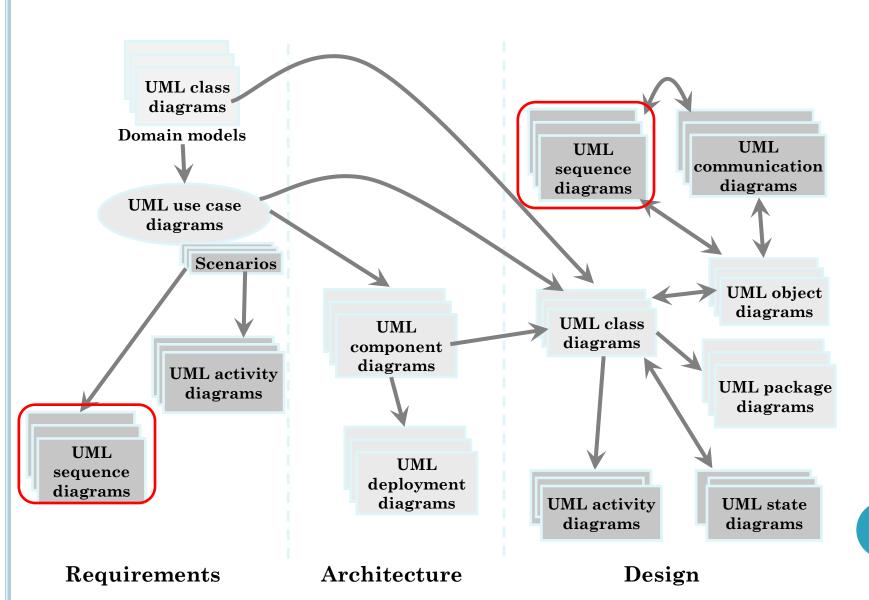


# UML 2.0: Sequence Diagram: Quando e dove usarli?

# QUANDO E DOVE È USATO UN SD?



### UML E PROCESSO



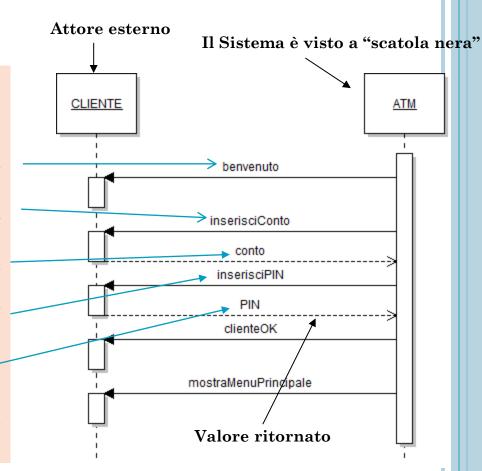
# SEQUENCE DIAGRAM DI SISTEMA (SSD)

• Un diagramma di sequenza di sistema illustra gli eventi di **input ed output del sistema** relativi ad uno **scenario** (o più) di un caso d'uso

#### **Autenticazione ATM**

**Primary Actor:** Cliente **Main Success Scenario:** 

- 1. Il Sistema visualizza un messaggio di benvenuto
- 2. Il sistema chiede al Cliente di inserire un numero di conto
- 3. Il Cliente digita un numero di conto di usando la tastiera
- 4. Il Sistema chiede al Cliente di inserire il PIN associato al conto
- 5. Il Cliente digita il PIN usando la tastiera
- 6. Il Sistema riconosce il cliente e visualizza il menu principale. Lo use case finisce con successo



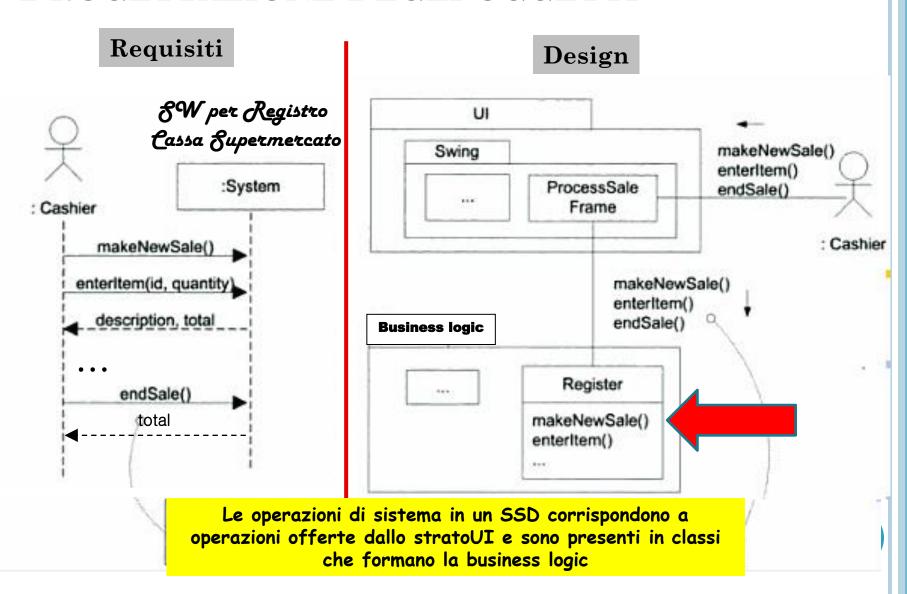
#### A COSA SERVE UN SSD?

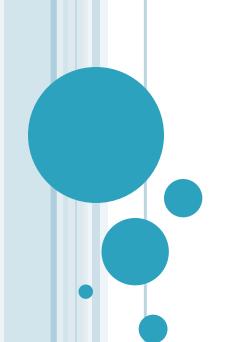
- Stabilire con precisione quali sono gli eventi di input (operazioni di input e parametri) e di output (dati di ritorno) del sistema
  - o Utile per chiarire input/output del sistema
  - o Utile nella fase di testing
- Rappresentare un punto di partenza per:
  - la progettazione delle classi e loro operazioni (Design)
  - l'analisi dei contratti delle operazioni

Come già detto è molto difficile 'passare' da Requisiti a Design!



### PROGETTAZIONE DEGLI OGGETTI

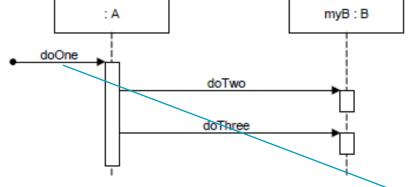




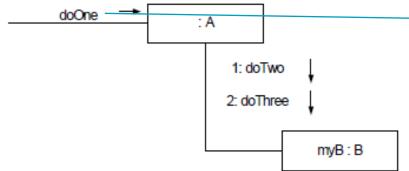
# UML 2.0: Communication Diagram

# DIAGRAMMI DI INTERAZIONE (ALMENO DUE TIPI)

Diagrammi di sequenza



Diagrammi di comunicazione



#### Una possibile implementazione della classe A in Java

public class A {

```
private B myB;

public A() {
  myB=new B();
}

public doOne() {
  myB.doTwo();
  myB.doThree();
}
```

## DIAGRAMMI DI INTERAZIONE (ALMENO DUE TIPI)

Diagrammi di sequenza

myB : B

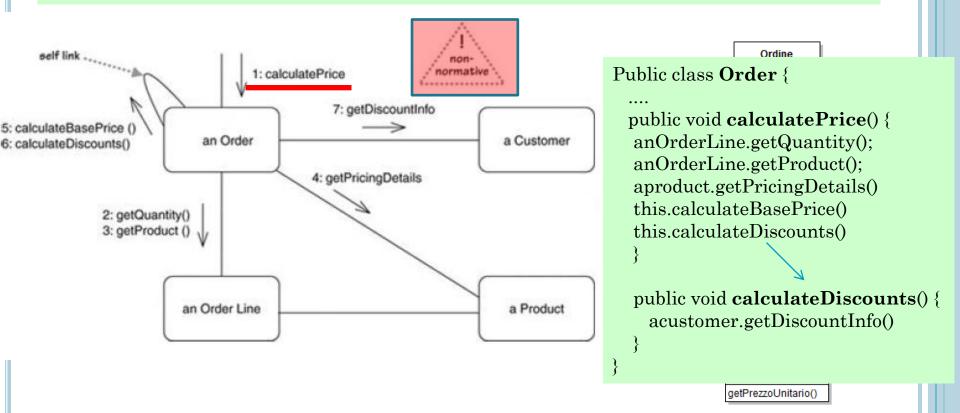
Diagrammi di sequenza e comunicazione forniscono la stessa informazione in modi diversi:

- i diagrammi di sequenza danno enfasi alla sequenza temporale dei messaggi scambiati
- i diagrami di comunicazione danno enfasi alla comunicazione tra oggetti

```
1: doTwo | myB.doTwo();
myB.doThree();
}
myB.doThree();
}
```

#### DIAGRAMMI DI COMUNICAZIONE

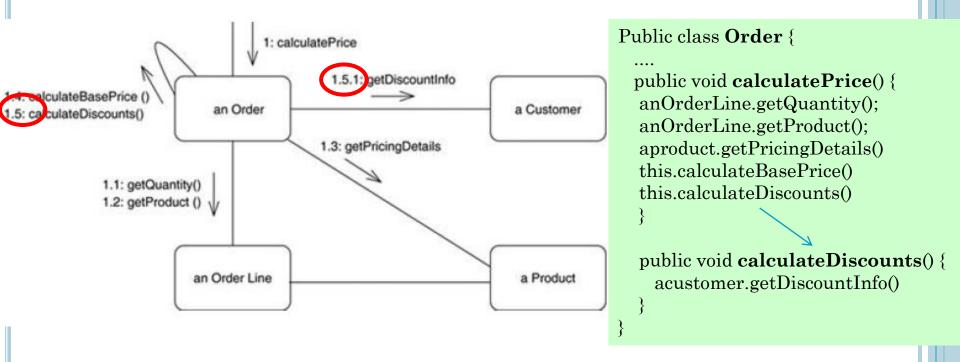
Rappresenta la sequenza di chiamate quando viene chiamato calculatePrice su un oggetto Order



Con questa numerazione non riesco a rappresentare le chiamate annidate!

Non si capisce che getDiscountInfo() è chiamata da calculateDiscounts()

#### Numerazione nidificata/progressiva



#### 1 calculatePrice()

- 1.1 anOrderLine.getQuantity();
- 1.2 anOrderLine.getproduct();
- 1.3 aproduct.getPricingDetails()
- 1.4 this.calculateBasePrice()
- 1.5 this.calculateDiscounts()
  - 1.5.1 acustomer.getDiscountInfo()

#### 'Dentro' a calculatePrice

'Dentro' a calculatePrice 'Dentro' a calculateDiscounts Warning: non possiamo vedere e non potete capire tutti i dettagli durante la lezione (seguite la registrazione dettagliata)



# ESERCIZIO: ORDINE DI ACQUISTO

- Supponiamo di avere un ordine di acquisto e di voler invocare un operazione per calcolare l'importo totale scontato
  - o.calcolaPrezzo()
- Per farlo l'ordine deve determinare l'importo di tutti i suoi **elementi di linea** 
  - Quantità \* prezzoUnitario
- Dopo avere calcolato il totale, l'ordine deve calcolare uno sconto globale che dipende dal cliente e scalarlo

#### Ordine di acquisto 'o'

ORDINE	DI ACQUISTO							
Azienda	ROAR spa							
Indirizzo	Via Grimaldello							
CAP	18017 Città S	an Lorenzo al Mare	Prov. IM					
Referente	Filippo Ricca	Tele						
	Prodotto	Prezzo	Quantità	Importo				
Prodotto A		€ 12,00	2	€ 24,0				
Prodotto B		€ 10,00	1	€ 10,0				
Prodotto C		€ 18,00	3	€ 54,0				
Prodotto D		€ 14,00	2	€ 28,0				
Prodotto E		€ 6,00	1	€ 6,0				
			Totale IMPORTO	£ 122 (				

€ 24,40

€ 146,40

## OVVERO ...

# Azienda ROAR spa Indirizzo Via Grimaldello CAP 18017 Città San Lorenzo al Mare Prov. IM Referente Filippo Ricca Telefono 338-3143\*\*\*

	Prodotto	Prezzo	Quantità	mporto
	Prodotto A	€ 12,00	2	€ 24,00
	Prodotto B	€ 10,00	1	€ 10,00
	Prodotto C	€ 18,00	3	€ 54,00
	Prodotto D	€ 14,00	2	€ 28,00
	Prodotto E	€ 6,00	1	€ 6,00
	-		Totale IMPORTO	€ 122,00
			IVA 20%	C 24,40
			TOTALE COMPLESSIVO	€ 146,40

o.calcolaPrezzo() = totale - sconto

#### Non teniamo conto dell'IVA!!!!

#### **ORDINE DI ACQUISTO**

#### Chi effettua l'ordine

Numero: 23

Cliente / Azienda

ROAR spa

Indirizzo

Via Grimaldello

CAP

18017

San Lorenzo al Mare Città

Prov. IM

Referente Filippo Ricca

Telefono 338-3143\*\*\*

Data dell'ordine

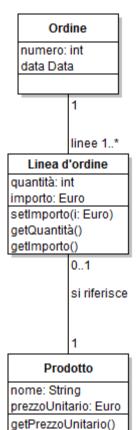
18/10/2011

Elementi di linea d'ordine

Prodotto	Prezzo	Quantità	Importo
Prodotto A	€ 12,00	2	€ 24,00
Prodotto B	€ 10,00	1	€ 10,00
Prodotto C	€ 18,00	3	€ 54,00
Prodotto D	€ 14,00	2	€ 28,00
Prodotto E	€ 6,00	1	€ 6,00
		Totale IMPORTO	€ 122,00
		IVA 20%	€ 24,40
		TOTALE COMPLESSIVO	€ 146,40

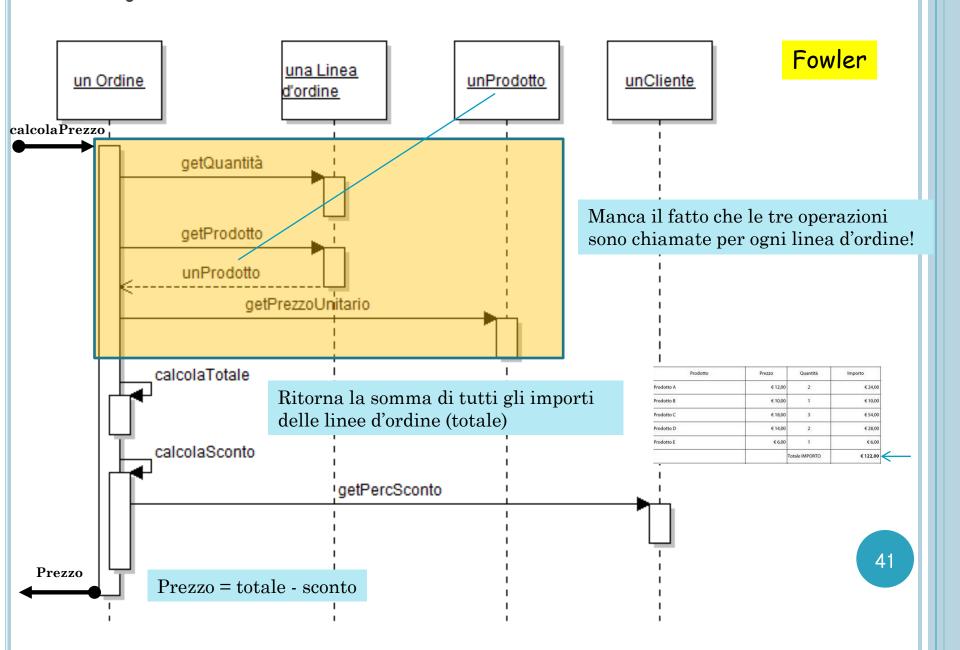
# QUALI CLASSI, ATTRIBUTI E OPERAZIONI?



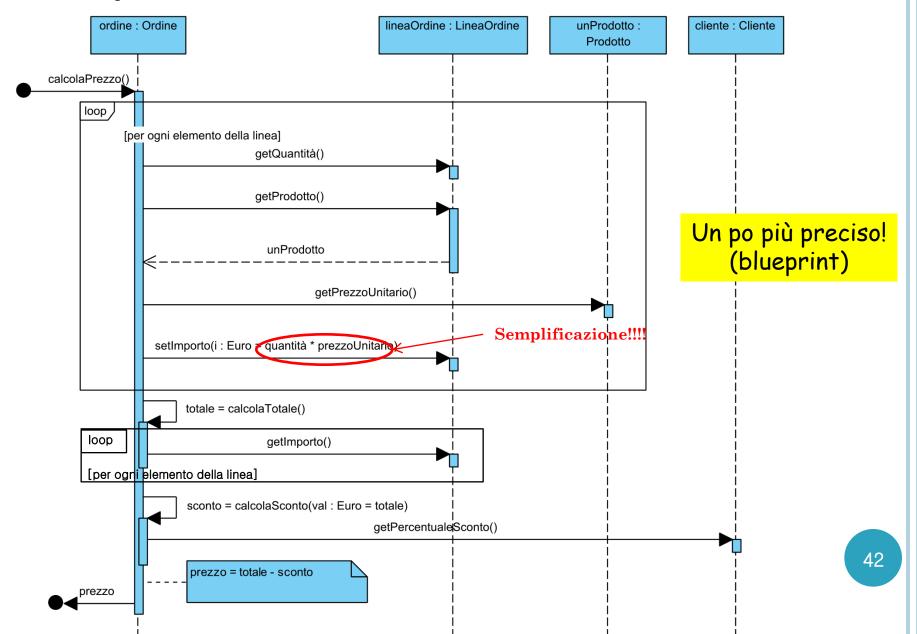




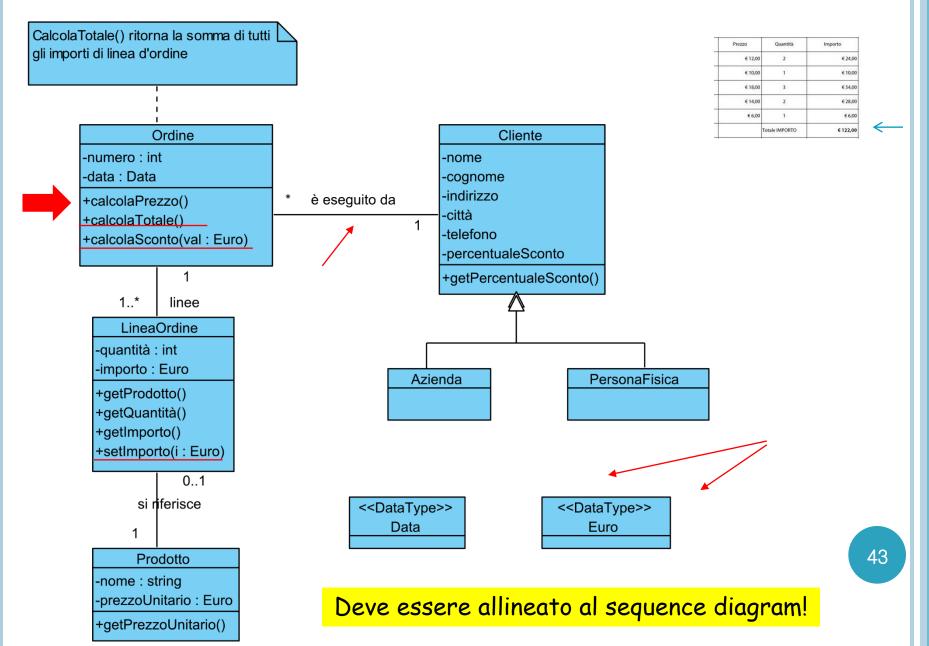
## SEQUENCE DIAGRAM: PRIMA SOLUZIONE



## SEQUENCE DIAGRAM: PRIMA SOLUZIONE



## CLASS DIAGRAM: PRIMA SOLUZIONE



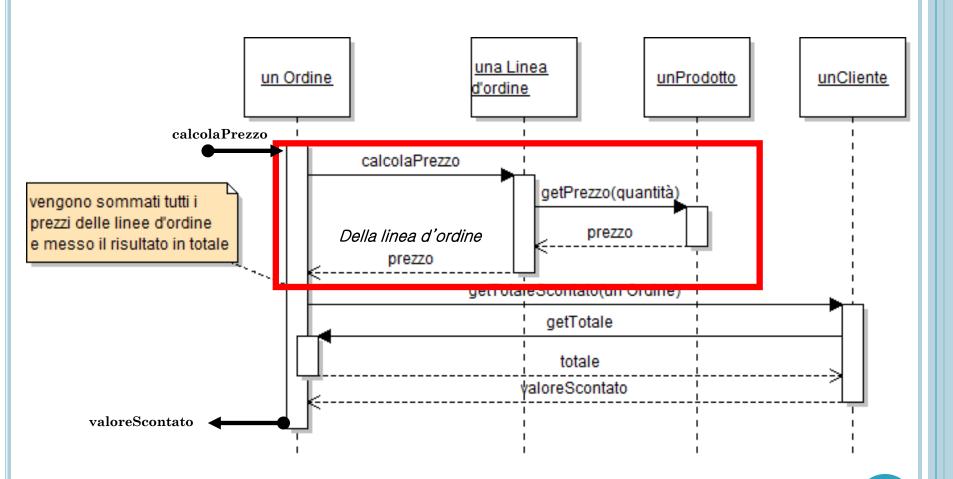
## PSEUDOCODICE: PRIMA SOLUZIONE

+, \* e - si assumono per Euro/int

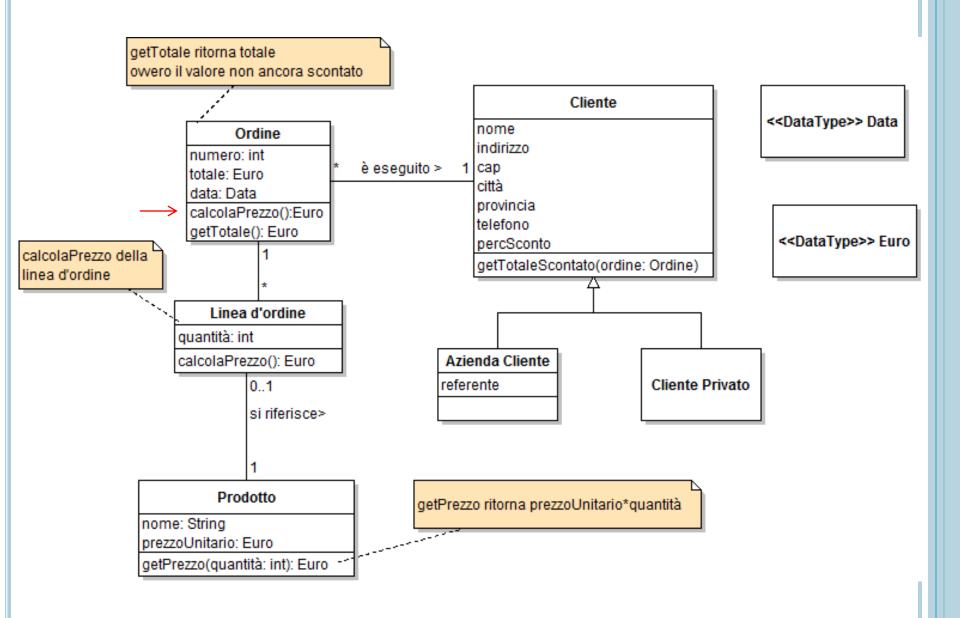
```
Public class Ordine {
 private HashSet<Linea> linee;
 private Cliente cliente;
 public Euro calcolaPrezzo() {
    int quantità;
    Prodotto p;
    Euro prezzoUnitario, sconto, totale;
    For (Linea linea: linee) {
       quantità = linea.getQuantità()
       p = linea.getProdotto();
       prezzoUnitario = p.getPrezzoUnitario()
       linea.setImporto(quantità*prezzoUnitario);
    totale = calcolaTotale():
    sconto = calcolaSconto(totale);
    return(totale - sconto)
```

```
private Euro calcolaTotale() {
      Euro tot=0:
      For (Linea linea: linee) {
          tot = tot + linea.getImporto();
    return tot;
  private Euro calcolaSconto(Euro t) {
        int perc = cliente.getPercSconto();
        return t*perc/100
                       Ordine
                -data : Data
                +calcolaPrezzo()
                +calcolaTotale()
                +calcolaSconto(val : Euro)
                         linee
                    LineaOrdine
                  -quantità : int
                  -importo : Euro
                 +getProdotto()
                 +getQuantità()
                 +getImporto()
                 +setImporto(i: Euro)
                      si riferisce
                                                           44
                      Prodotto
                  -nome : string
                  prezzoUnitario : Euro
                  getPrezzoUnitario()
```

## SEQUENCE DIAGRAM: SECONDA SOLUZIONE



#### CLASS DIAGRAM: SECONDA SOLUZIONE



#### PSEUDOCODICE: SECONDA SOLUZIONE

+, \* e - si assumono per Euro/int

```
Public class Ordine {
 private HashSet<Linea> linee;
 private Cliente cliente;
 private Euro totale;
 public Euro calcolaPrezzo() {
    Euro prezzo, sconto;
    totale = 0;
    For (Linea linea: linee) {
       prezzo = linea.calcolaPrezzo();
       totale = totale +prezzo;
    return(cliente.getTotaleScontato(this))
 public Euro getTotale() {
    return totale;
```

```
private Prodotto prodotto;
    private int quantità;
    public Euro calcolaPrezzo() {
      return prodotto.getPrezzo(quantità):
Public class Cliente {
 private int percSconto;
public Euro getTotaleScontato(Ordine o) {
    Euro totale = o.getTotale()
    Euro sconto = totale*percSconto/100;
    return totale – sconto;
```

Public class **Linea** {

#### **CONFRONTO**

- Nella prima soluzione un partecipante (oggetto ordine) svolge tutta l'elaborazione mentre gli altri forniscono solo i dati
  - Controllo centralizzato
- Nella seconda soluzione invece ogni partecipante contribuisce in parte all'elaborazione dell'algoritmo
  - Controllo distribuito



Quale delle due soluzioni pensate sia la migliore?

#### LA SOLUZIONE DISTRIBUITA ...

#### • Due motivi:

- Dal momento che i dati e la logica che li gestisce cambiano spesso contemporaneamente, raccoglierli nello stesso posto riduce il numero di classi da modificare
  - Tra l'altro raccogliere **dati + operazioni nella stessa classe** è una delle prime regole dell'OO

#### • Limita le God class

- God class o Blob class (difficili da capire e modificare)
  - Classi che gestiscono tutta la logica del software
  - Possibili sintomi:
    - Classe complessa (CC media della classe alta)
    - Classe poco coesa (LCOM alta)
  - Infatti nella prima soluzione la classe **Ordine** è una God class

# THE END ...



Domande?