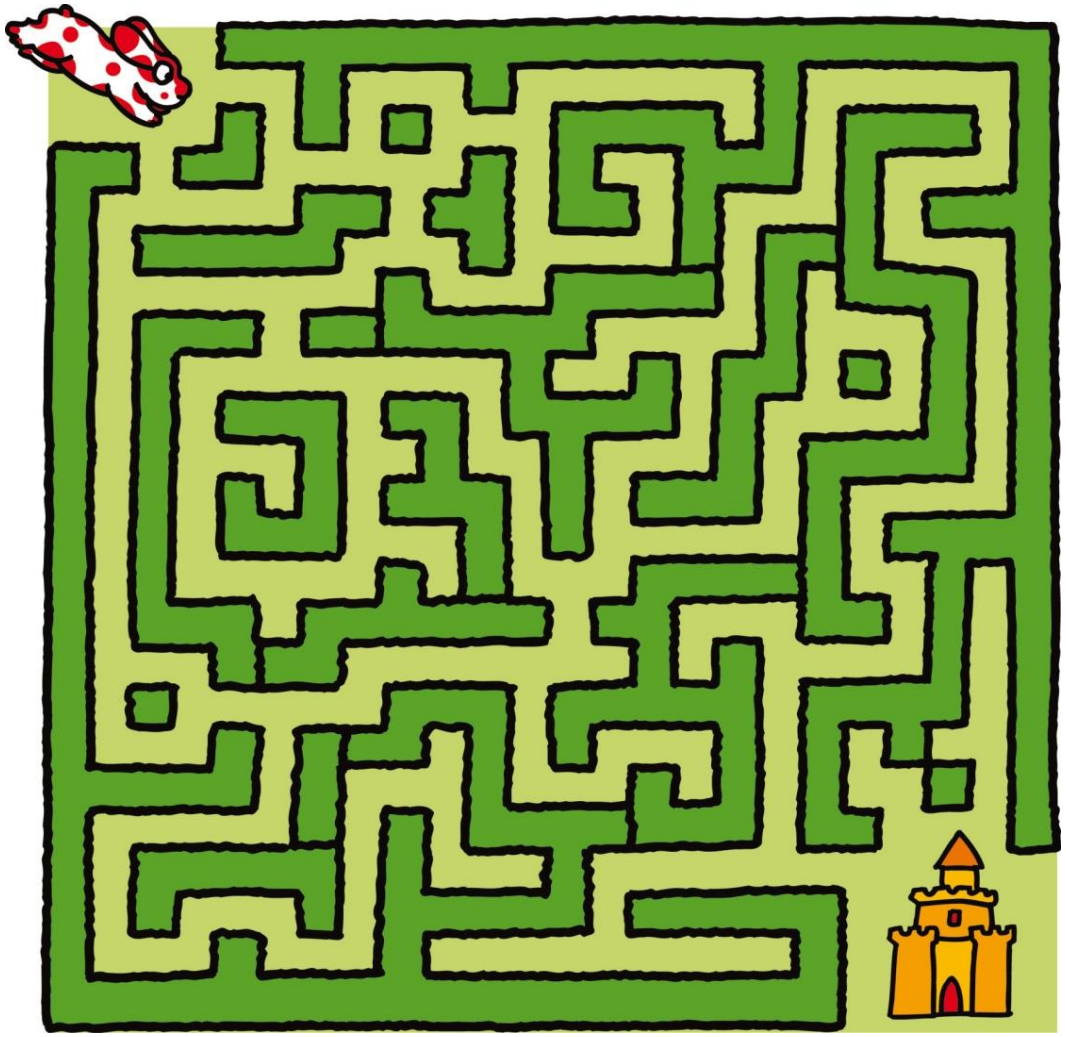
UML 2.0: SEQUENCE DIAGRAM
(+ COMMUNICATION DIAGRAM)

Ingegneria del Software 2023-2024

Orientiamoci …



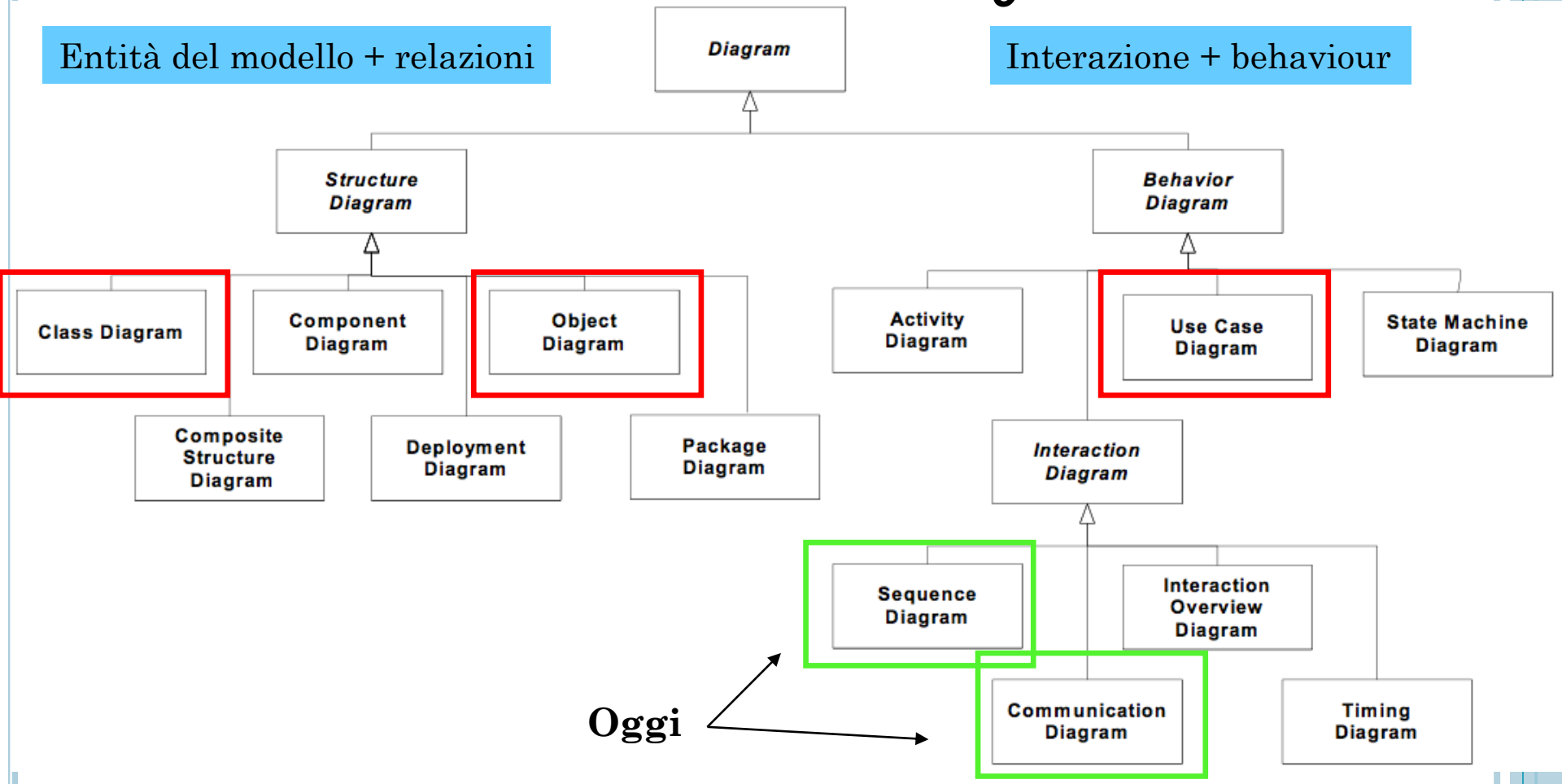
ORIENTIAMOCI …

Modellazione statica

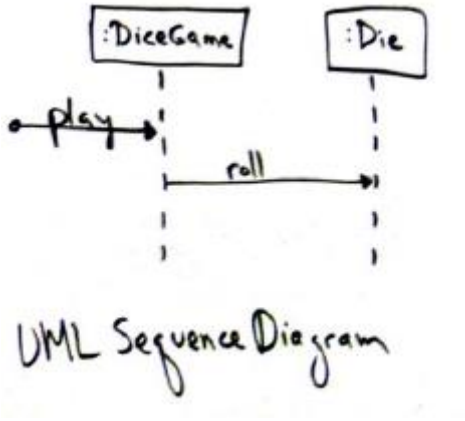
Entità del modello + relazioni

Modellazione dinamica

Interazione + behaviour



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**

ORDINE DI ACQUISTO

Azienda ROAR spa

Indirizzo Via Grimaldello

CAP 18017 Città San Lorenzo al Mare Prov. IM

Referente Filippo Ricca Telefono 338-3143***

Data dell'ordine 18/10/2011

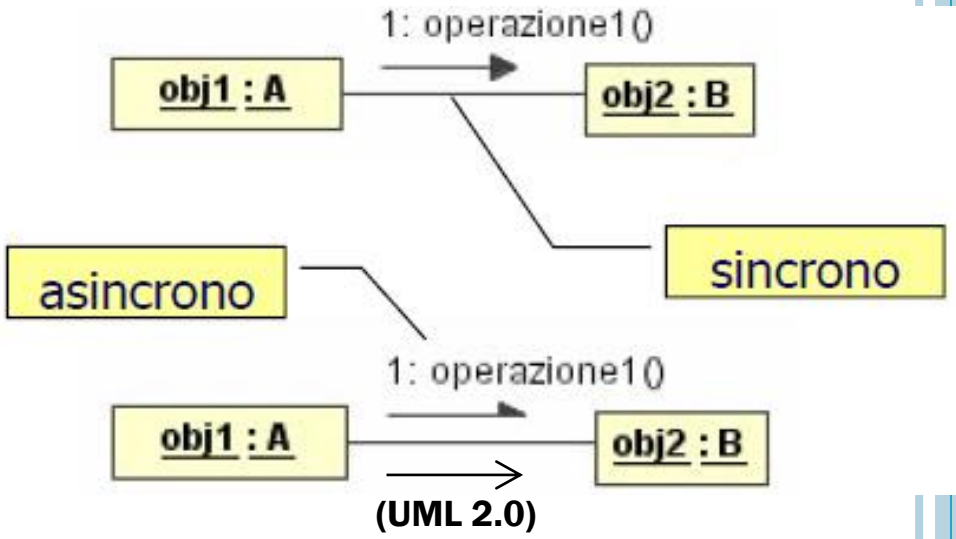
Prodotto	Prezzo	Quantità	Importo
Prodotto A	€ 12,00	2	€ 24,00
Prodotto B	€ 10,00	1	€ 10,00
Prodotto C	€ 18,00	3	€ 54,00
Prodotto D	€ 14,00	2	€ 28,00
Prodotto E	€ 6,00	1	€ 6,00
		Totale IMPORTO	**€ 122,00**
		IVA 20%	**€ 24,40**
		TOTALE COMPLESSIVO	**€ 146,40**

Diagrammi di interazione

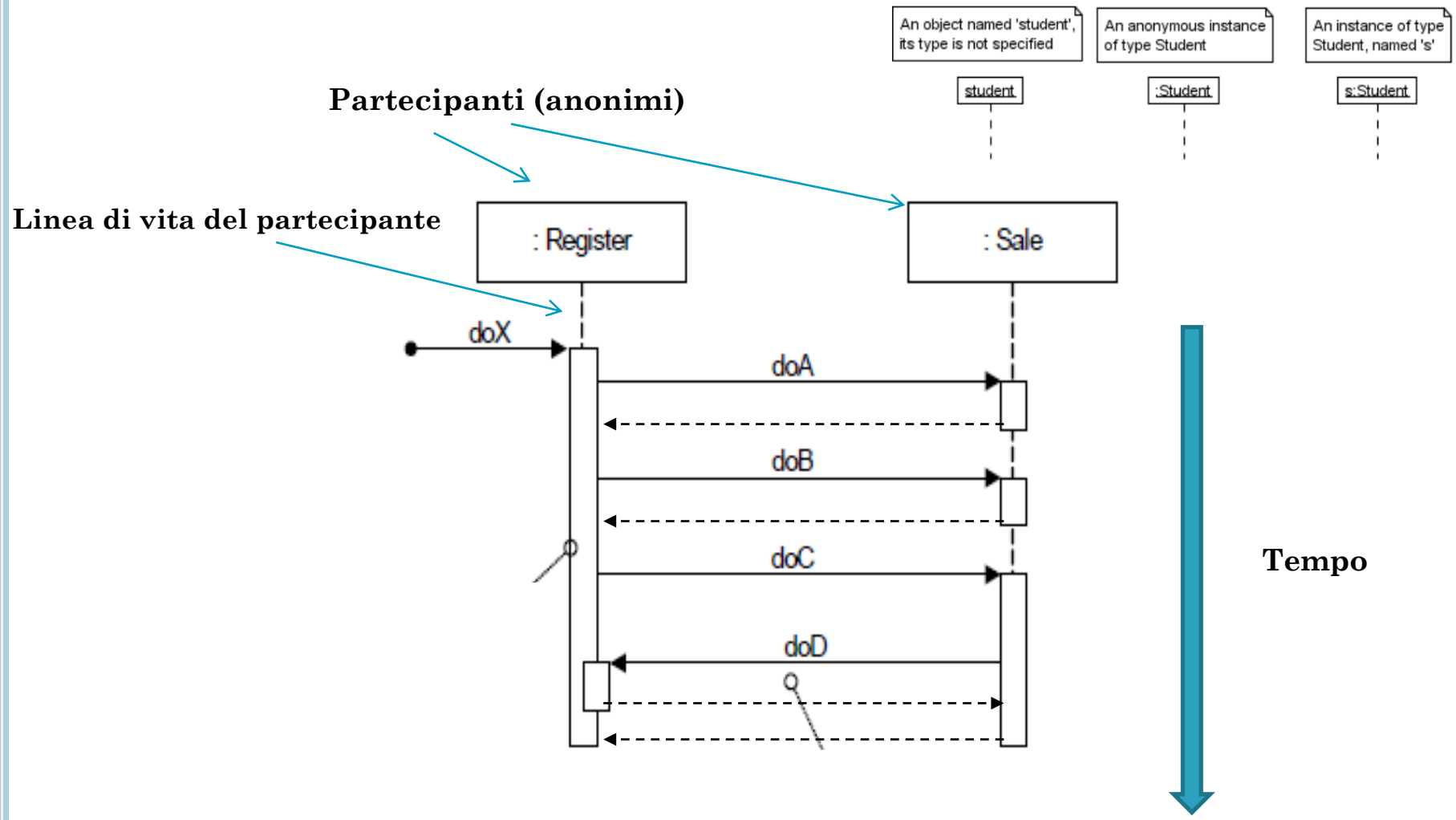
- I **diagrammi di interazione** descrivono **la collaborazione** di un gruppo di oggetti
  - ovvero lo scambio di messaggi …
- I messaggi possono essere
  - **sincroni**: il mittente si pone in attesa del risultato
  - **asincroni** il mittente continua l'esecuzione
    - usati per rappresentare i processi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

Per mandare un msg deve esistere una relazione (es. associazione o attributo)

Messaggio = chiamata di operazione

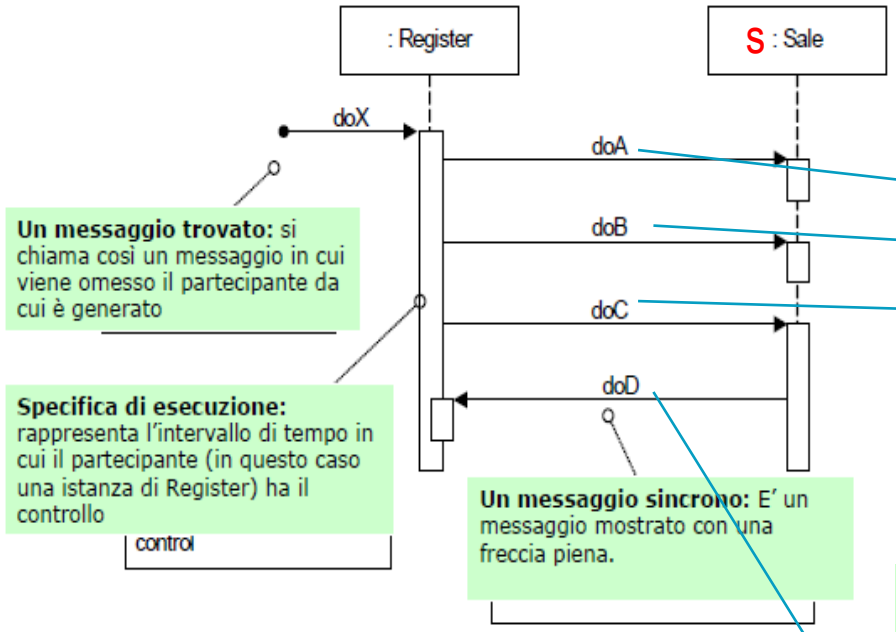


Notazione Sequence Diagram



Esegue un'operazione o ha passato il controllo ad un'altra operazione

POSSIBILE IMPLEMENTAZIONE




```
Public class Register {
   private Sale s;
   public Register() {
      s = new Sale();
    }
   public void doX() {
      s.doA()
      s.doB()
      s.doC(this)
    }
   public void doD() { ... }
}
```

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

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

d = getProduct(id)

SINTASSI DEI MESSAGGI

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

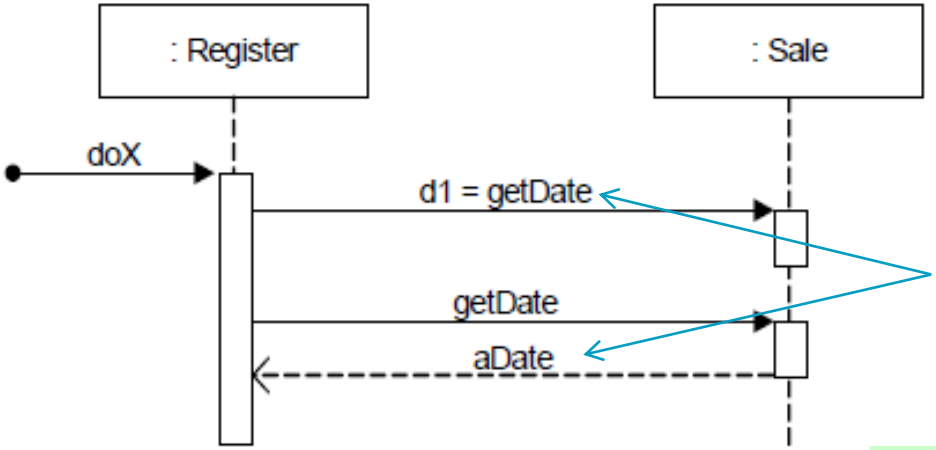
parameters è una lista di 'parameter:parameterType' separati da virgola

Esempio:

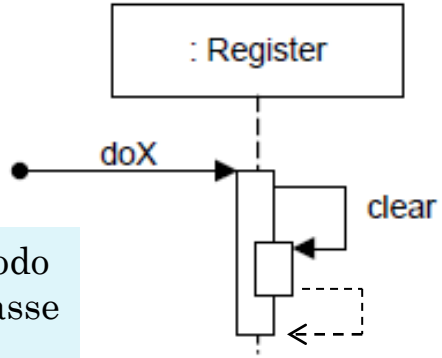
- getProduct
- d = getProduct
- d = getProduct(id)
- d = getProduct(id:ItemID)
- d = getProduct(id:ItemID) : Product

Tutti ammessi!

VALORI DI RITORNO E MESSAGGI SELF



Valori di ritorno
due modi equivalenti!!!

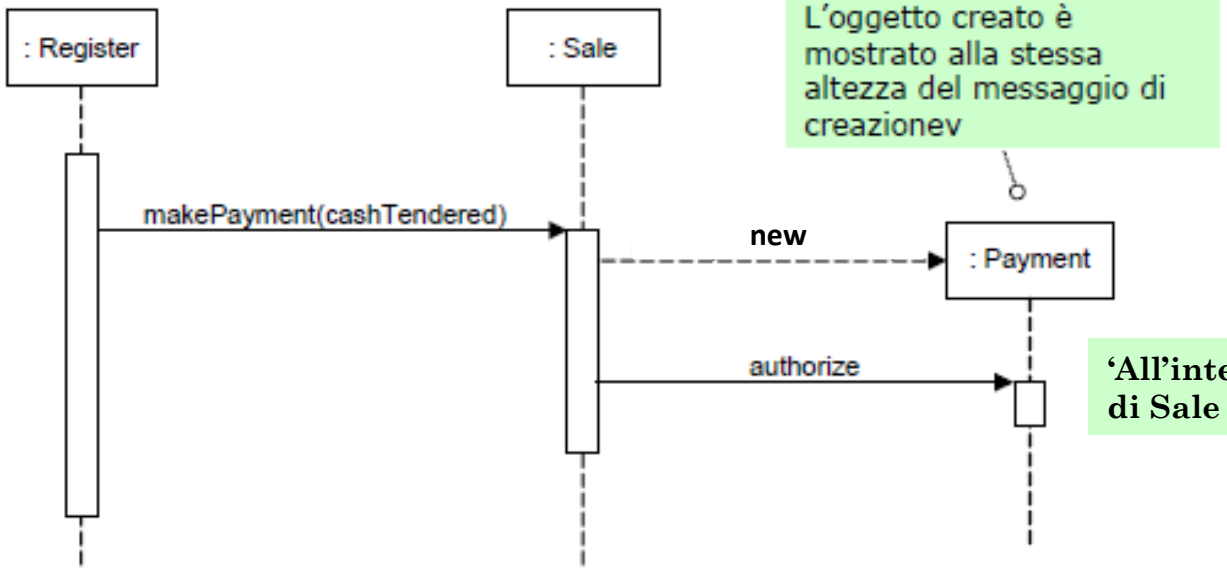


Self

(chiamata di un metodo appartenente alla classe stessa)

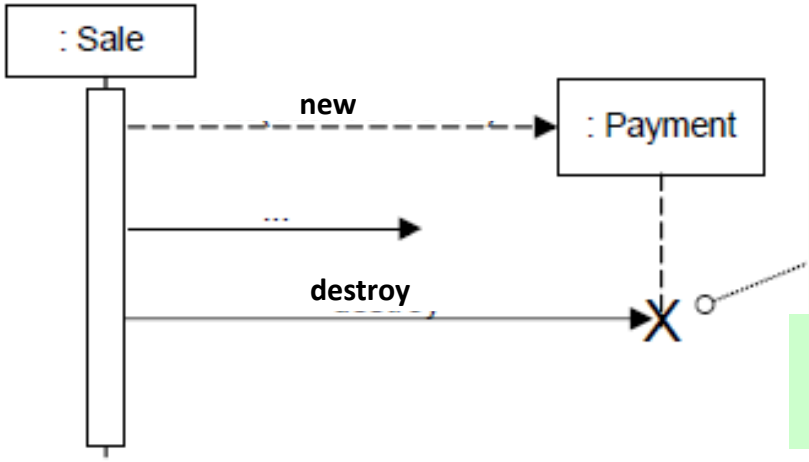
```
Public class Register {
    public void doX() {
        this.clear()
        ...
    }
    public void clear() {
    ....
    }
}
```

Creazione e distruzione



L'oggetto creato è mostrato alla stessa altezza del messaggio di creazionev

'All'interno dell'operazione makePayment() di Sale abbiamo il comando "new Payment()"



Una X sulla linea di vita rappresenta un messaggio di distruzione esplicita

Negli ambienti dotati di garbage collector La X significa che l'oggetto non serve più e può essere cancellato ...

Un'oggetto si può anche autodistruggere

Esempio: ATM

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

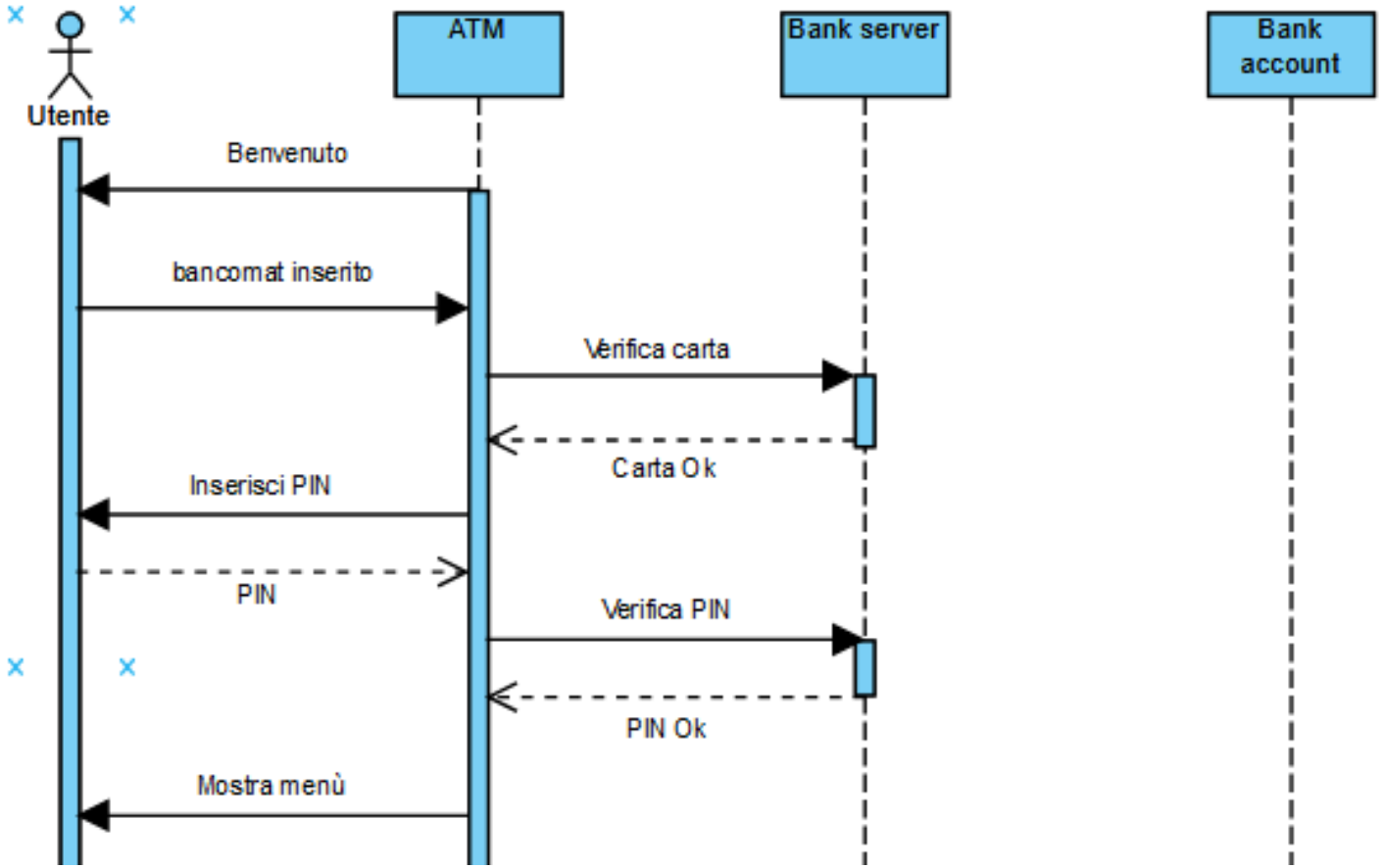
Utente
(è un partecipante)

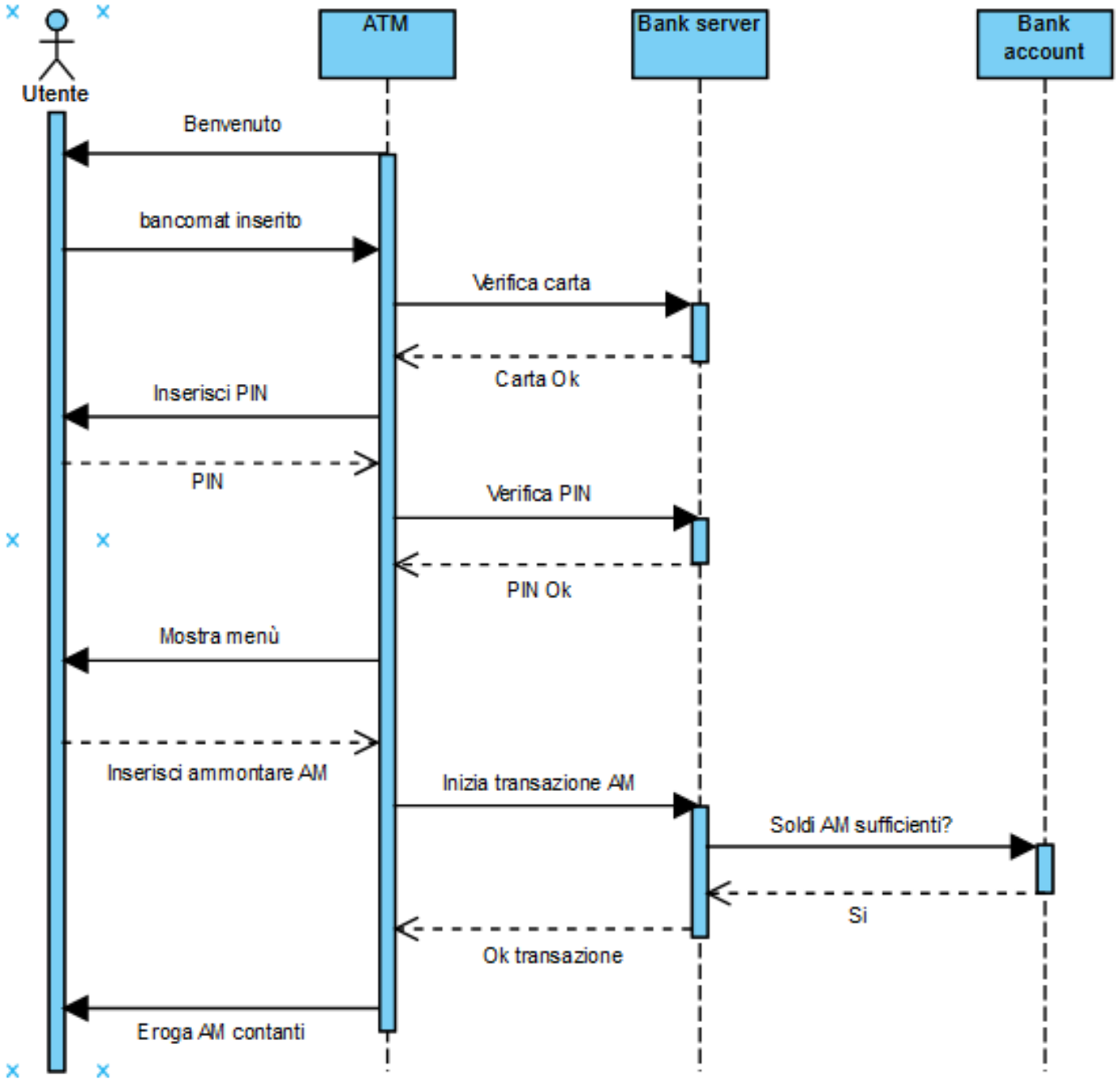
ATM

Bank server

Bank account

Sistema SW


Utente
ATM
Bank server
Bank account
Benvenuto
bancomat inserito
Verifica carta
Carta Ok
Inserisci PIN
PIN
Verifica PIN
PIN Ok
Mostra menù


Utente
ATM
Bank server
Bank account
Benvenuto
bancomat inserito
Verifica carta
Carta Ok
Inserisci PIN
PIN
Verifica PIN
PIN Ok
Mostra menù
Inserisci ammontare AM
Inizia transazione AM
Soldi AM sufficienti?
Si
Ok transazione
Eroga AM contanti

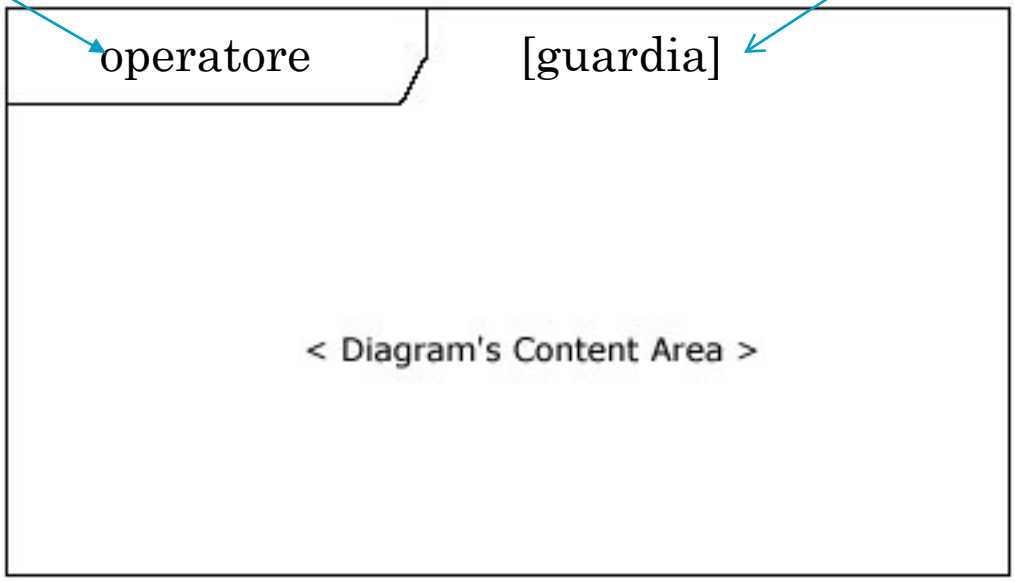
UML 2.0:
Sequence Diagram: Frame

CICLI CONDIZIONI E ALTRO ....

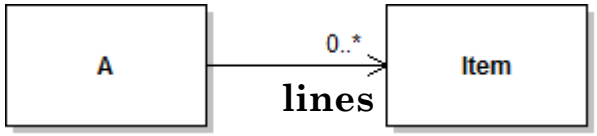
- **Cicli** e **condizioni** (la logica di controllo) possono essere espressi mediante i **frame di interazione**
  - Una sorta di "cornice" che serve ad inquadrare una parte del diagramma

Esistono diversi operatori

Opzionale (dipende dal tipo di operatore)



FRAME (1)

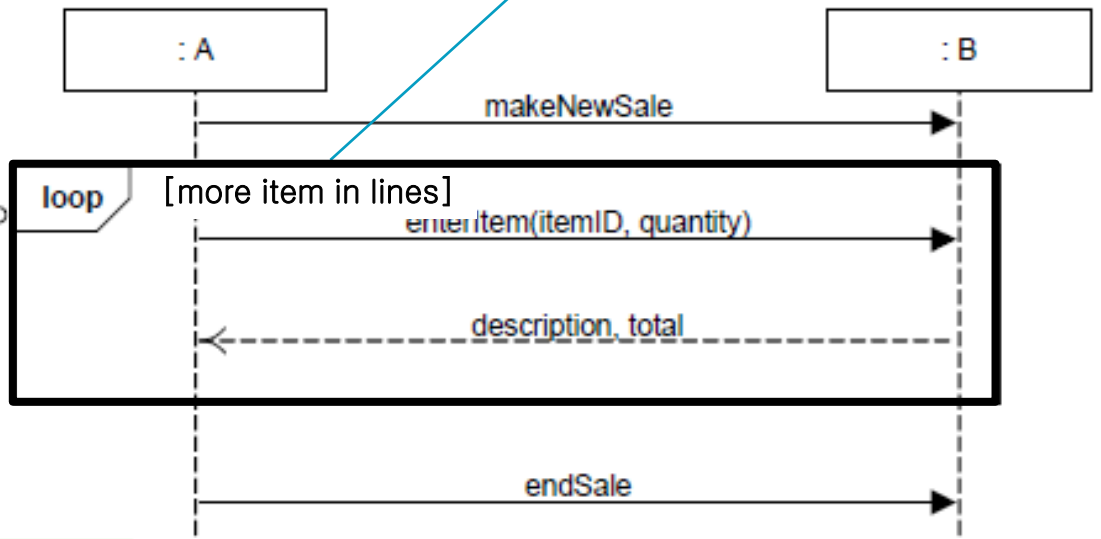


For (Item item: lines)
b.enterItem(item.itemID, item.quantity)

Per ogni item in lines, un oggetto di tipo A manda un msg enterItem(...) ad un oggetto di tipo B

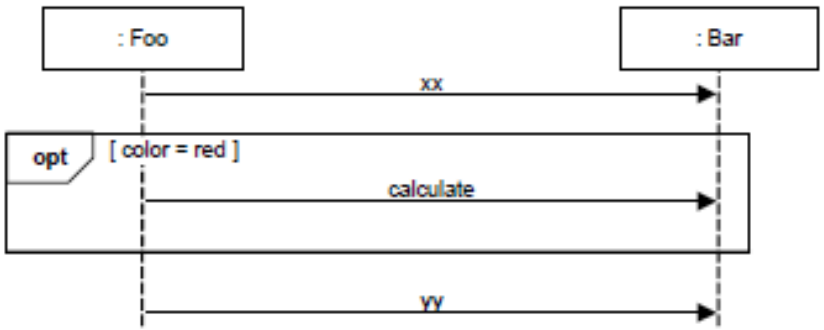
Loop

Ciclo con un espressione di guardia booleana

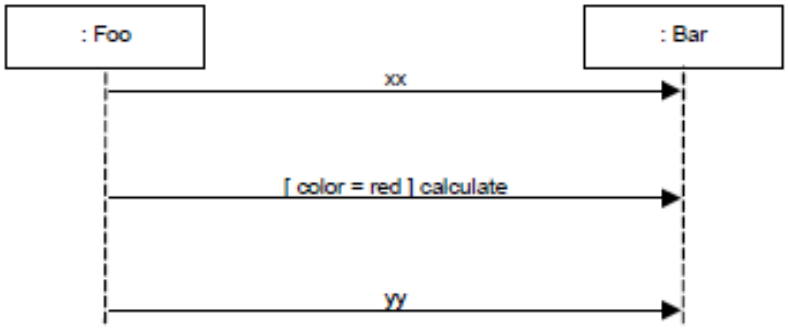


Opt

Condizione con UML 2.0



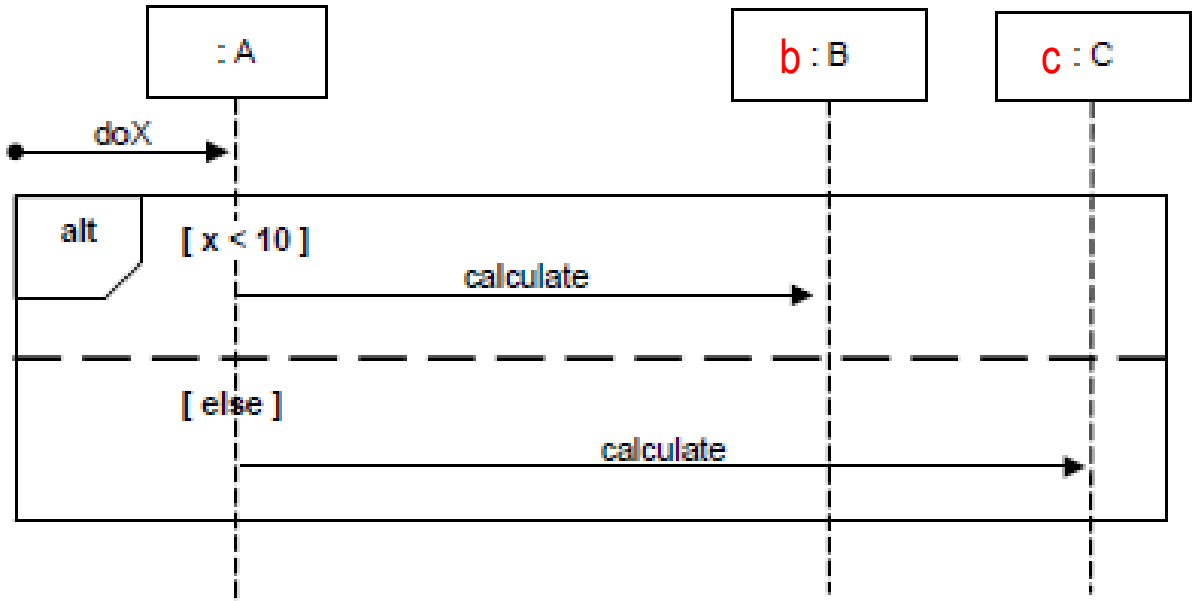
Condizione con UML 1.0



Il frame viene eseguito solo se la condizione è valida

FRAME (2): IF THEN ELSE / CASE

Alt

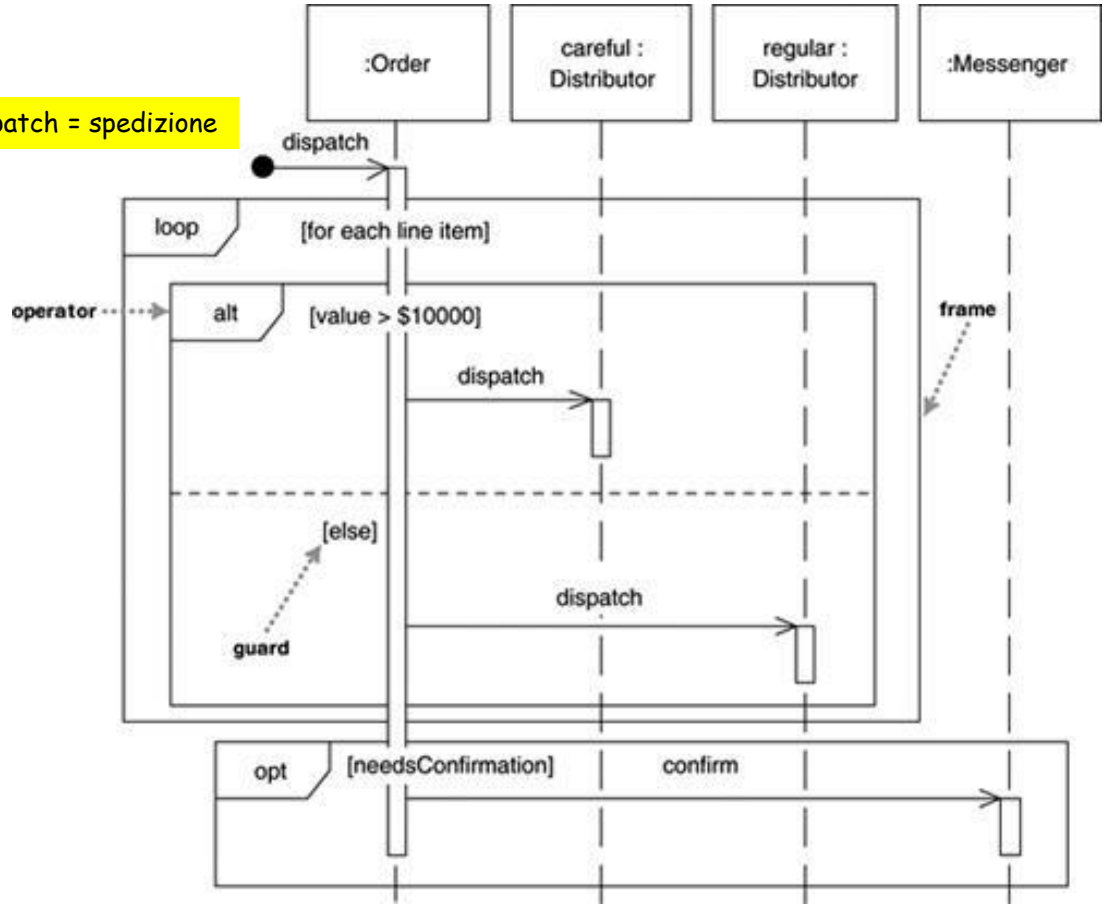


Si possono aggiungere più alternative (corrisponde al Case)

```
Public class A {
   private B b;
   private C c;
   private int x;

   public A() {
      b = new B();
      c = new C();
      x = …
    }

   public void doX() {
      if x < 10  {
        b.calculate()
     }
       else{
        c.calculate()
     }
    }

  …
}
```

Frame (3): Frame annidati

Dispatch = spedizione



ORDINE DI ACQUISTO

Azienda ROAR spa

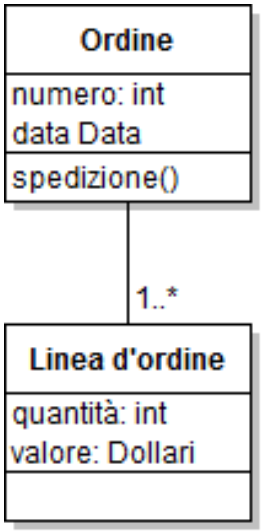
Indirizzo Via Grimaldello

CAP 18017 Città San Lorenzo al Mare Prov. IM

Referente Filippo Ricca Telefono 338-3143***

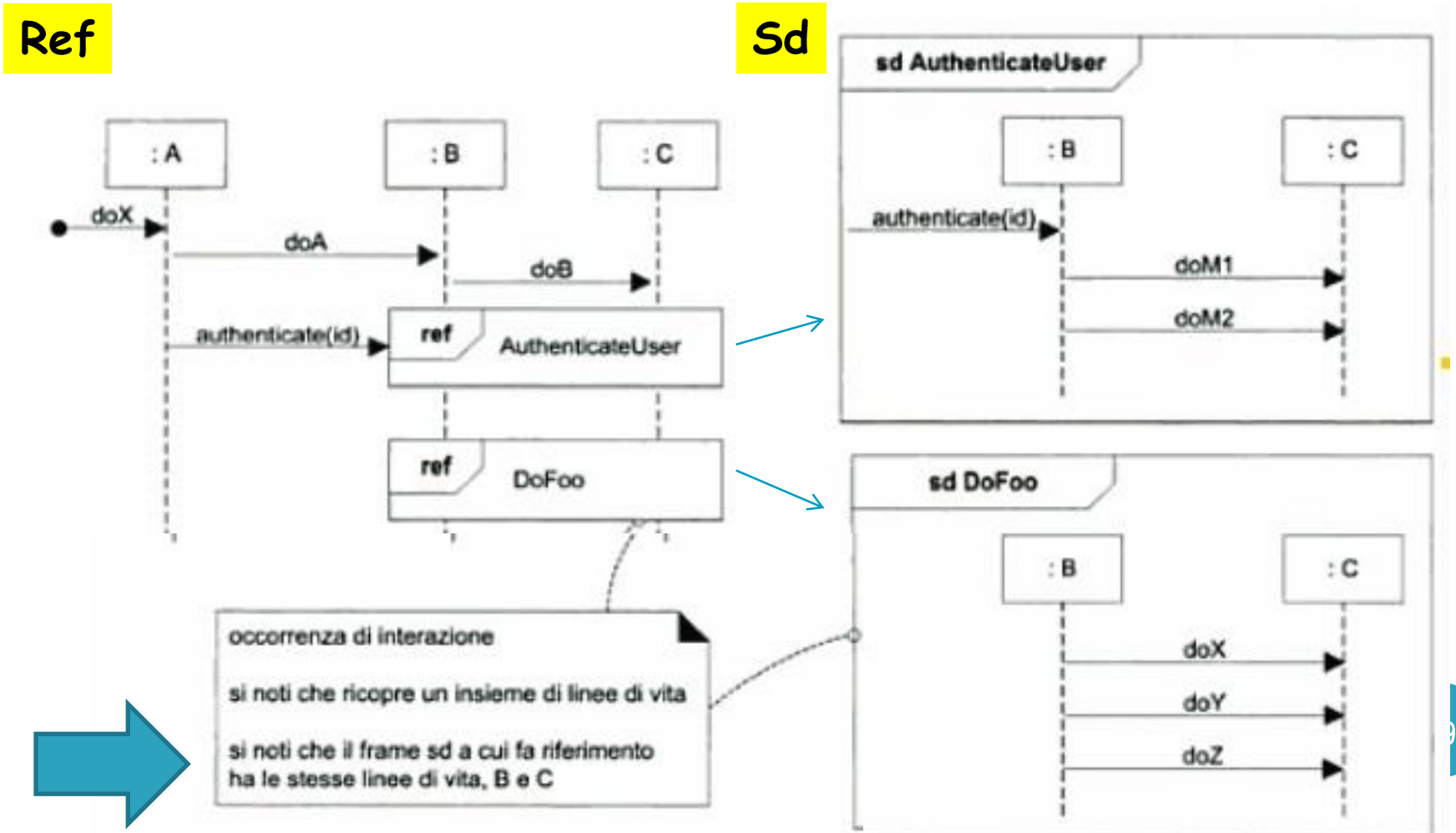
Data dell'ordine 18/10/2011

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**



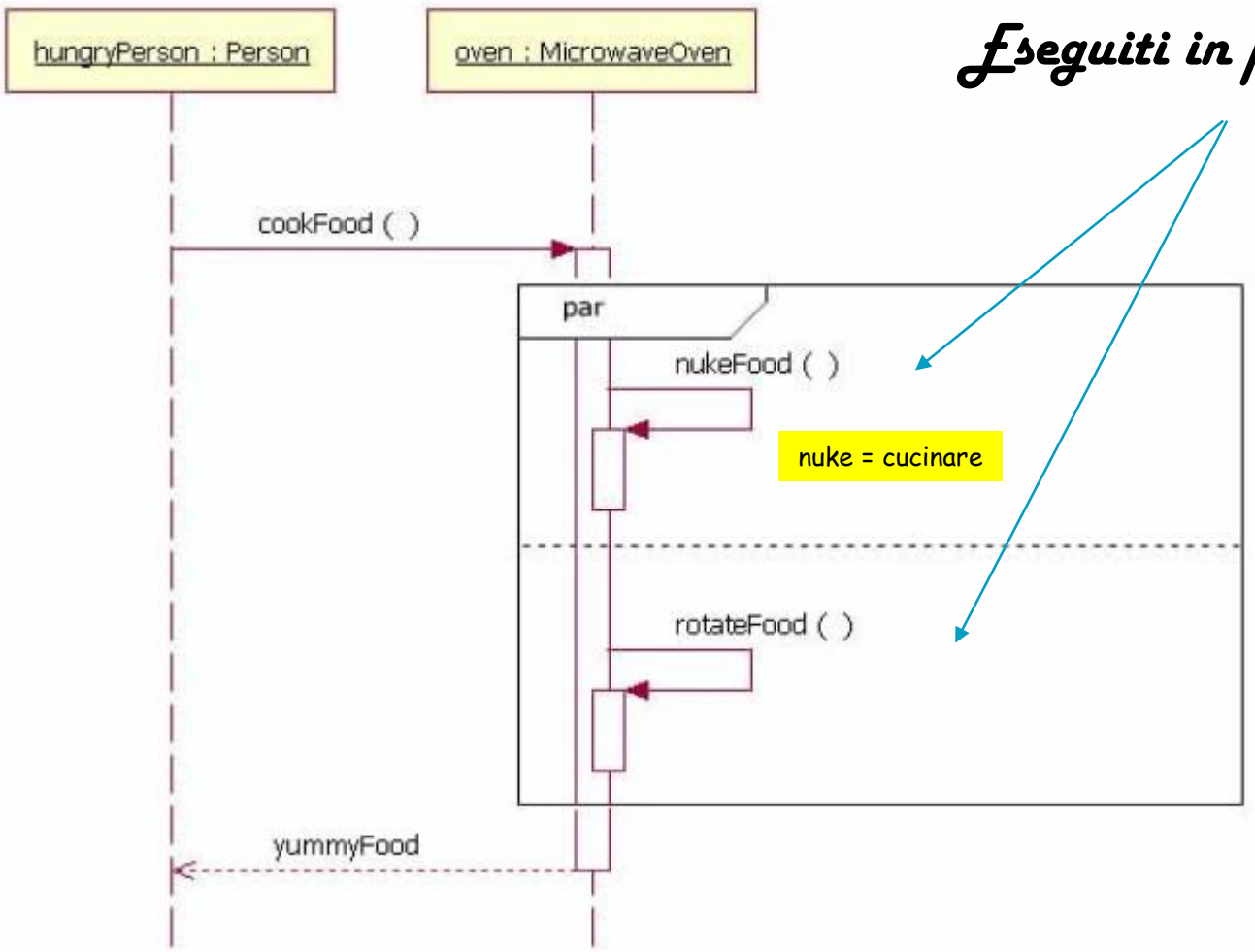
Frame (4): Ref

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



Frame (5): Par

Par



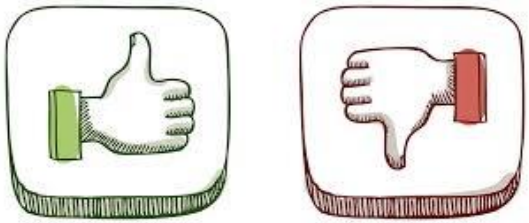
Utile per esprimere l'interazione di oggetti complessi che possono eseguire diverse azioni/operazioni in parallelo (es. Forno a microonde)

Operatori usati con i frame

`alt`	Alternative multiple fragments; only the one whose condition is true will execute (Figure 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).
`par`	Parallel; each fragment is run in parallel.
`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.
`sd`	Sequence diagram; used to surround an entire sequence diagram, if you wish.

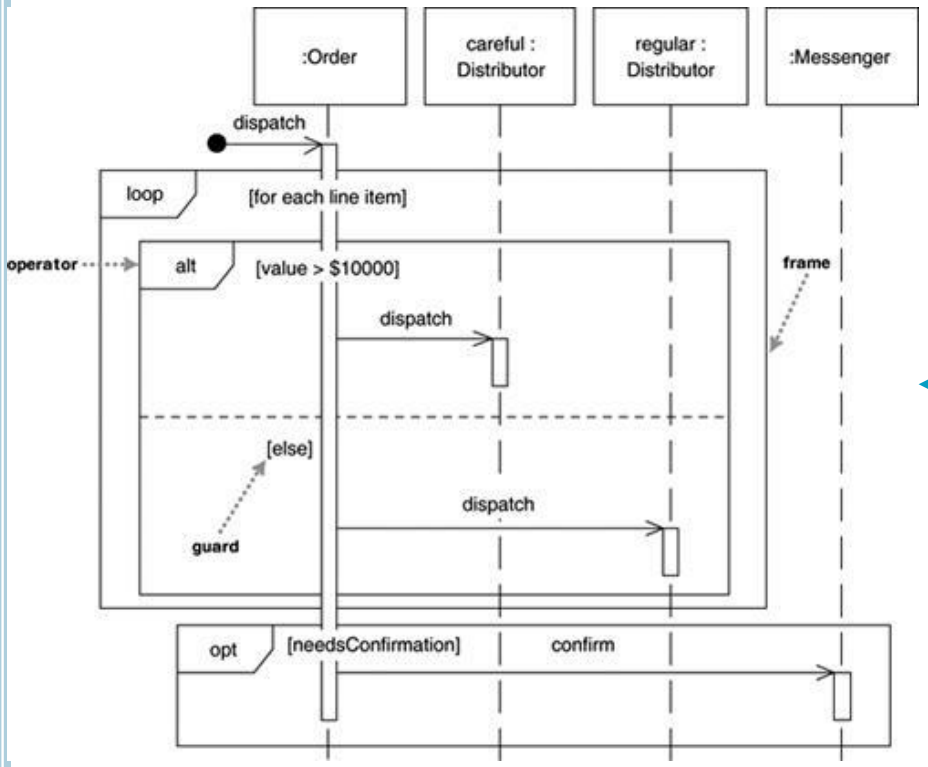
Figura 4.4 del libro UML distilled

Pro e Contro



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



Sequence diagram

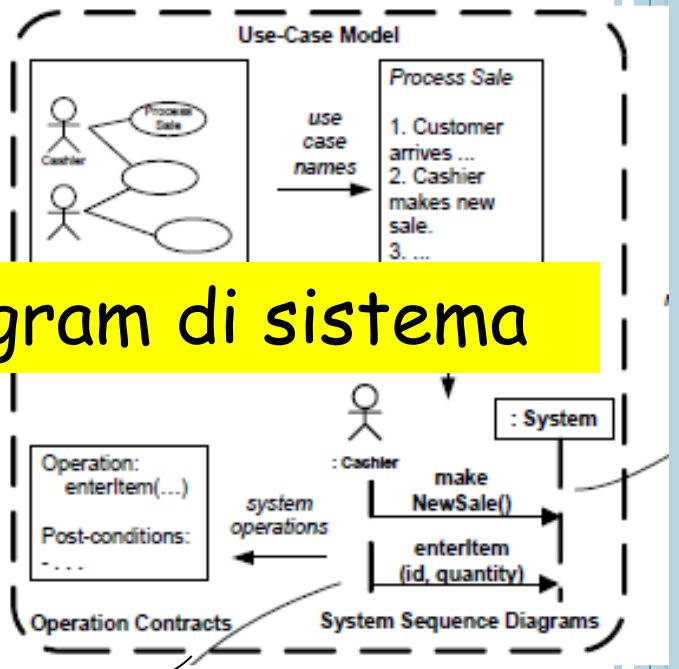
Pseudocodice

```
procedure 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?

- **A due livelli differenti**:
  - **Usati a livello di Requisiti**
    - Use case diagram
    - Descrizione d
    - Diagrammi d
      - **Attori e sistema**
    - Contratti delle operazioni
  - **Usati a livello di Design**



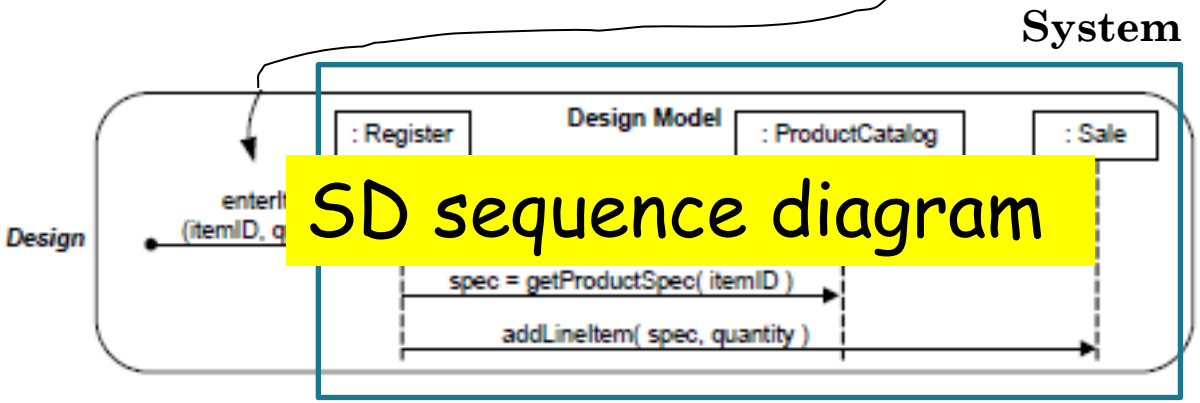
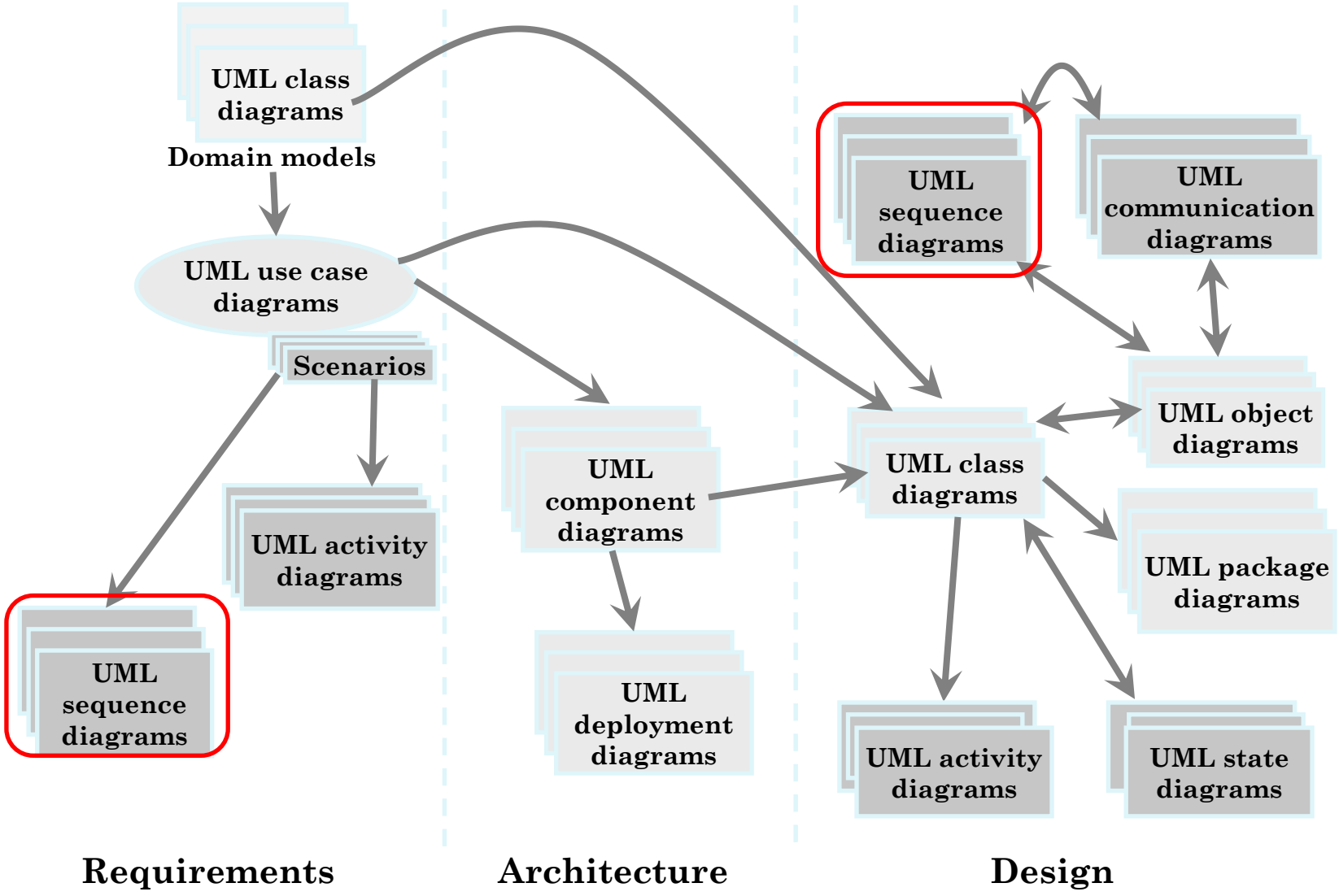


Immagine presa da Pfleeger (modificata)

UML E PROCESSO



Sequence Diagram di sistema (SSD)

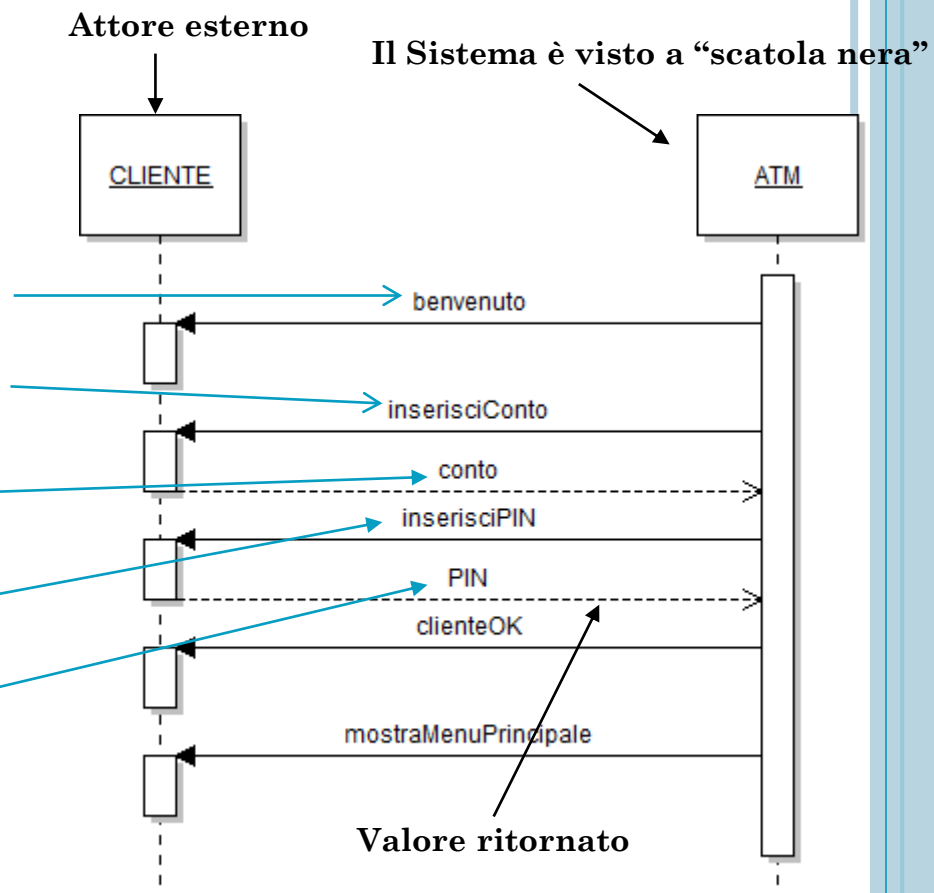
- Un diagramma di sequenza di sistema illustra gli eventi di **input ed output del sistema** relativi ad uno **scenario** (o più) di un caso d'uso

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?

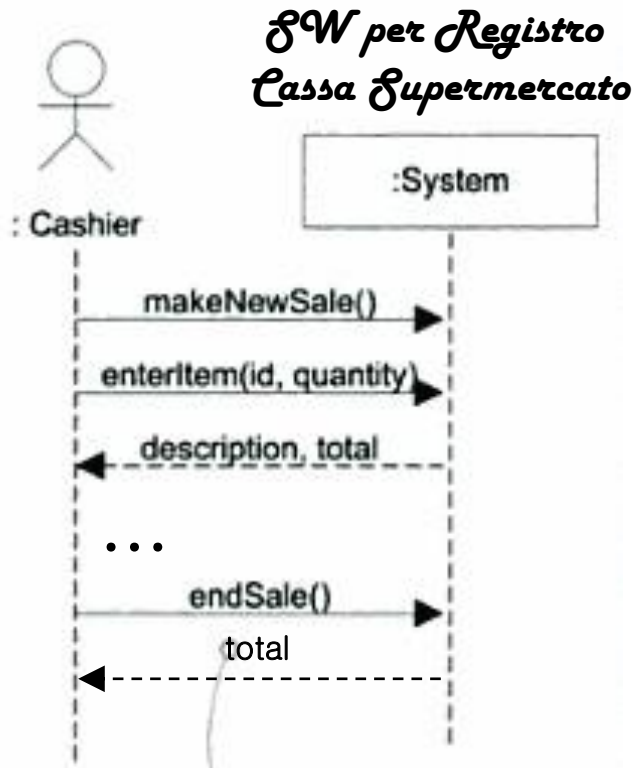
- Stabilire con precisione quali sono gli eventi di input (operazioni di input e parametri) e di output (dati di ritorno) del sistema
  - **Utile per chiarire input/output del sistema**
  - **Utile nella fase di testing**
- Rappresentare un punto di partenza per:
  - la **progettazione delle classi e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

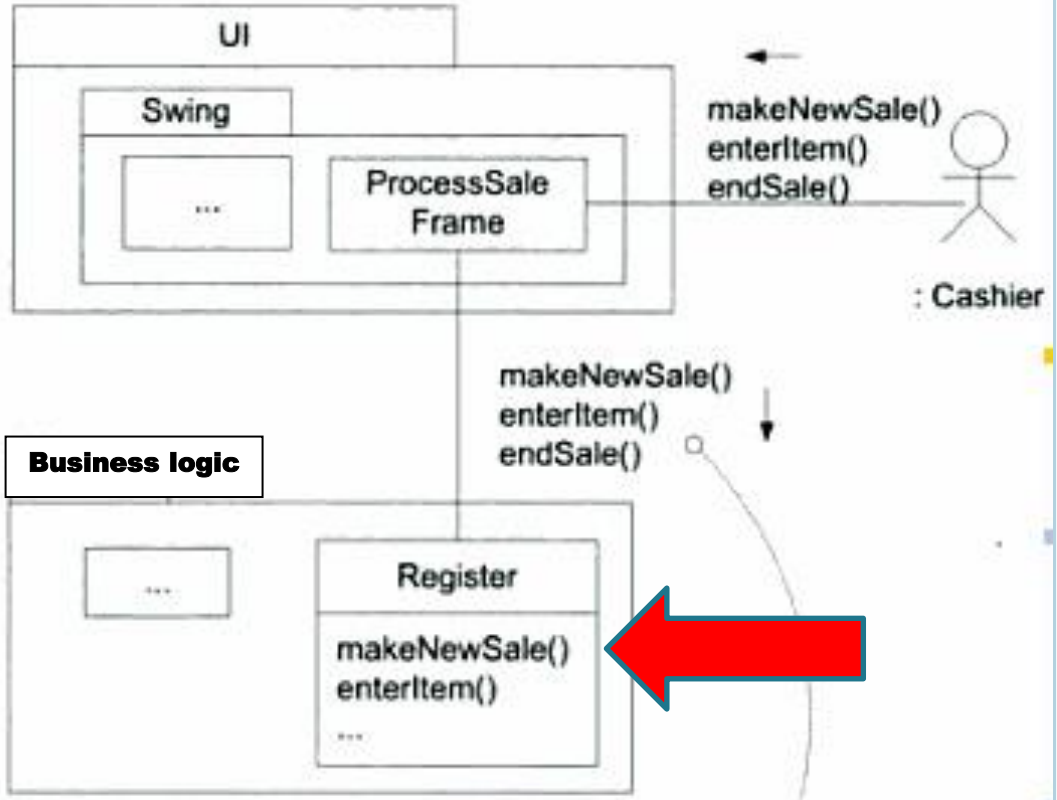
Come già detto è molto difficile 'passare' da Requisiti a Design!

PROGETTAZIONE DEGLI OGGETTI

Requisiti

Design



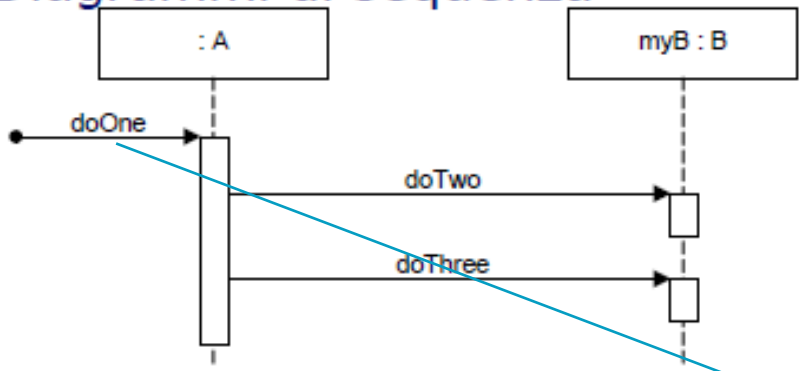


Le operazioni di sistema in un SSD corrispondono a operazioni offerte dallo stratoUI e sono presenti in classi che formano la business logic

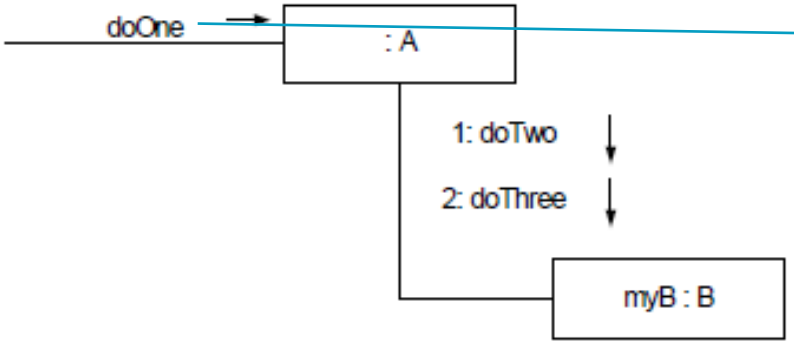
UML 2.0: Communication Diagram

Diagrammi di interazione (almeno due tipi)

- Diagrammi di sequenza



- Diagrammi di comunicazione

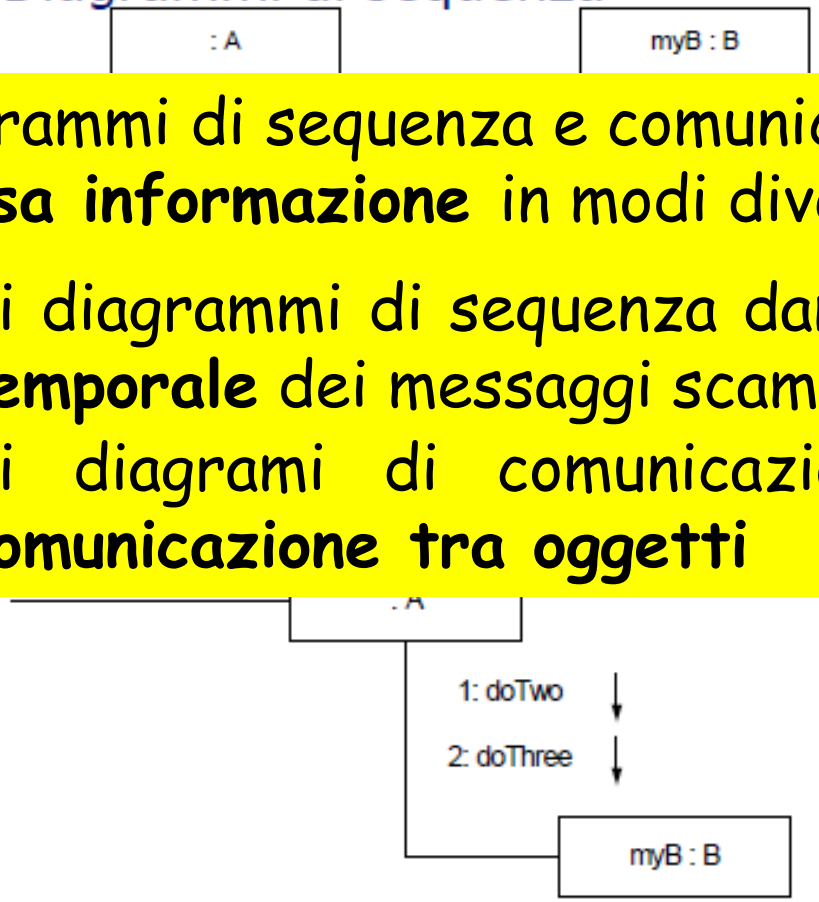


Una possibile implementazione della classe A in Java

```
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



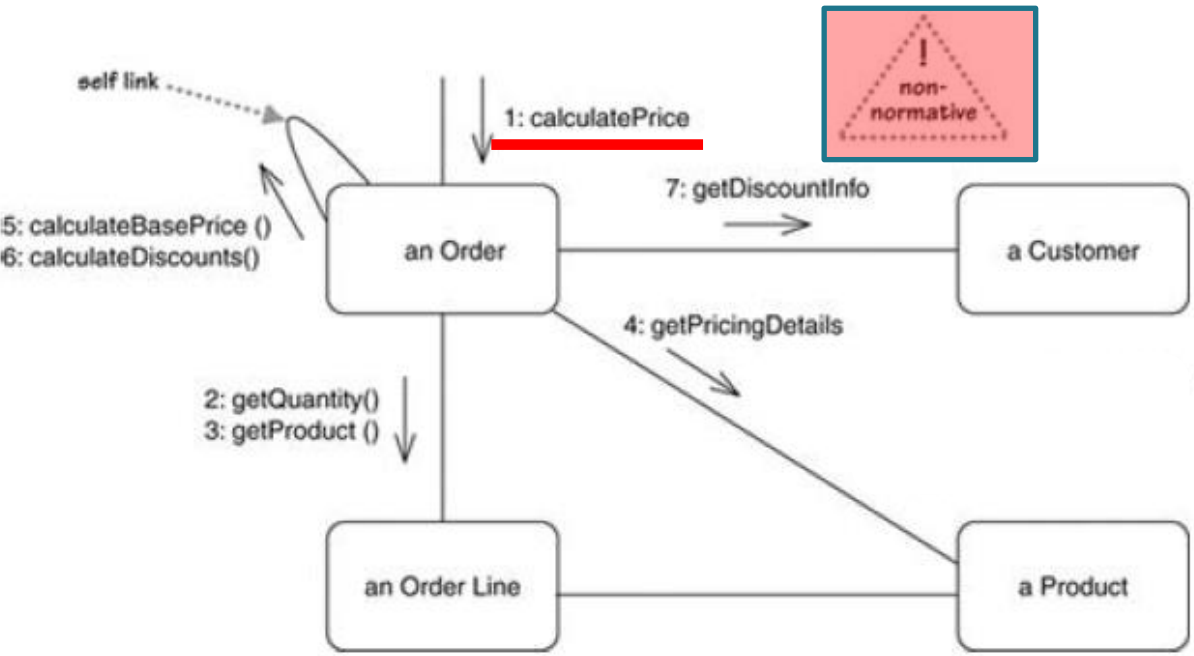
Diagrammi di sequenza e comunicazione forniscono la **stessa informazione** in modi diversi:

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

```
myB.doTwo();
myB.doThree();
}
}
```

DIAGRAMMI DI COMUNICAZIONE

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



Ordine

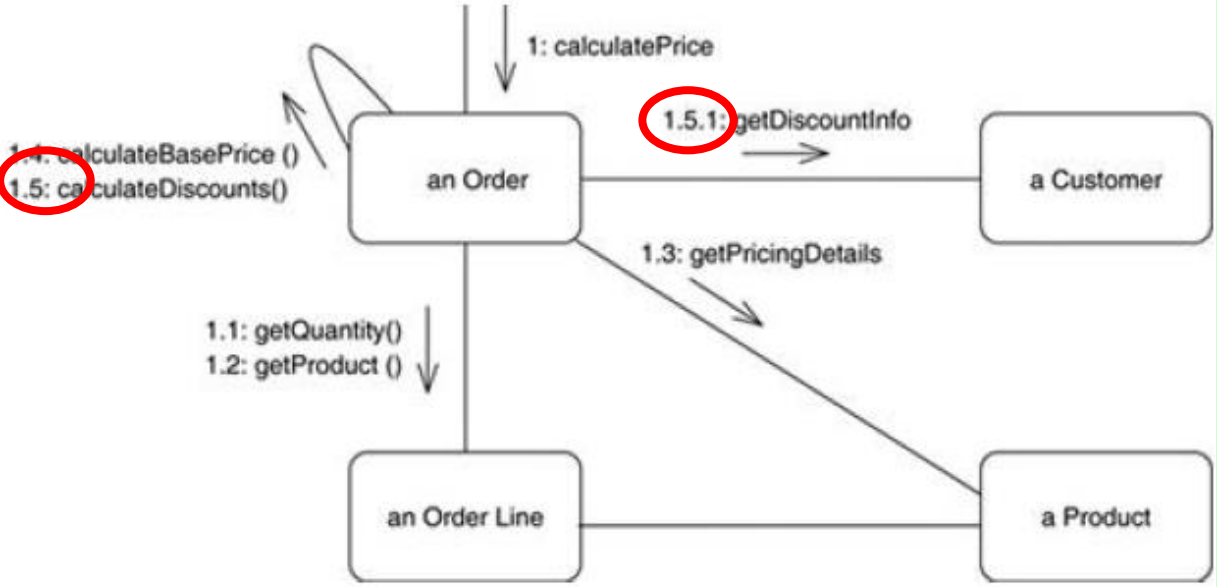
```
Public class Order {
   ....
   public void calculatePrice() {
    anOrderLine.getQuantity();
    anOrderLine.getProduct();
    aproduct.getPricingDetails()
    this.calculateBasePrice()
    this.calculateDiscounts()
    }

    public void calculateDiscounts() {
      acustomer.getDiscountInfo()
    }
}
```

getPrezzoUnitario()

Con questa numerazione non riesco a rappresentare le chiamate annidate!

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

NUMERAZIONE NIDIFICATA/PROGRESSIVA




```
Public class Order {
   ....
   public void calculatePrice() {
    anOrderLine.getQuantity();
    anOrderLine.getProduct();
    aproduct.getPricingDetails()
    this.calculateBasePrice()
    this.calculateDiscounts()
    }

    public void calculateDiscounts() {
       acustomer.getDiscountInfo()
    }
}
```

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 (li vedrete e capirete in laboratorio)

ESERCIZIO (Ordine)

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 San Lorenzo al Mare Prov. IM

Referente Filippo Ricca Telefono 338-3143***

Data dell'ordine 18/10/2011

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**

OVVERO …

ORDINE DI ACQUISTO

Azienda ROAR spa

Indirizzo Via Grimaldello

CAP 18017 Città San Lorenzo al Mare Prov. IM

Referente Filippo Ricca Telefono 338-3143***

Data dell'ordine 18/10/2011

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%	
		TOTALE COMPLESSIVO	**€ 146,40**

Elementi di linea d'ordine

o.calcolaPrezzo() = totale – sconto

= 24+10+54+28+6 = 122
Se lo sconto per 'ROAR' = 10%
= 122-12.2 = 109.8

Non teniamo conto dell'IVA!!!!

ORDINE DI ACQUISTO

Chi effettua l'ordine

Numero: 23

Cliente / Azienda ROAR spa

Indirizzo Via Grimaldello

CAP 18017 Città San Lorenzo al Mare 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?

ORDINE DI ACQUISTO

Numero: 23

Azienda ROAR spa

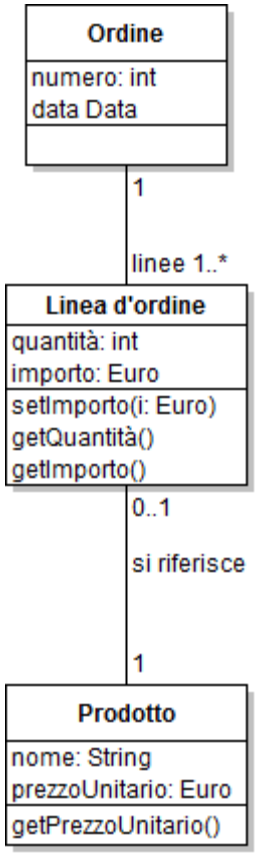
Indirizzo Via Grimaldello

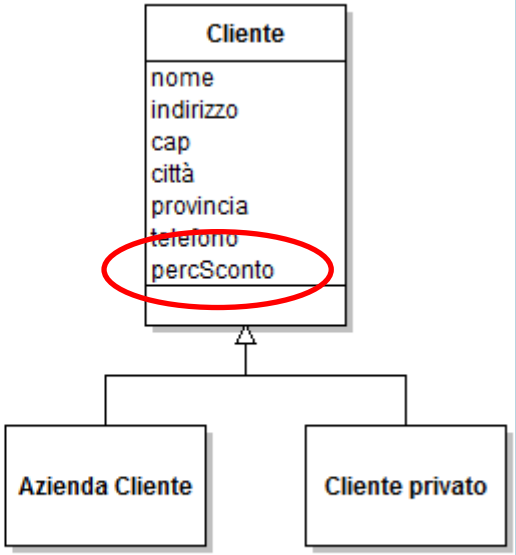
CAP 18017 Città San Lorenzo al Mare Prov. IM

Referente Filippo Ricca Telefono 338-3143***

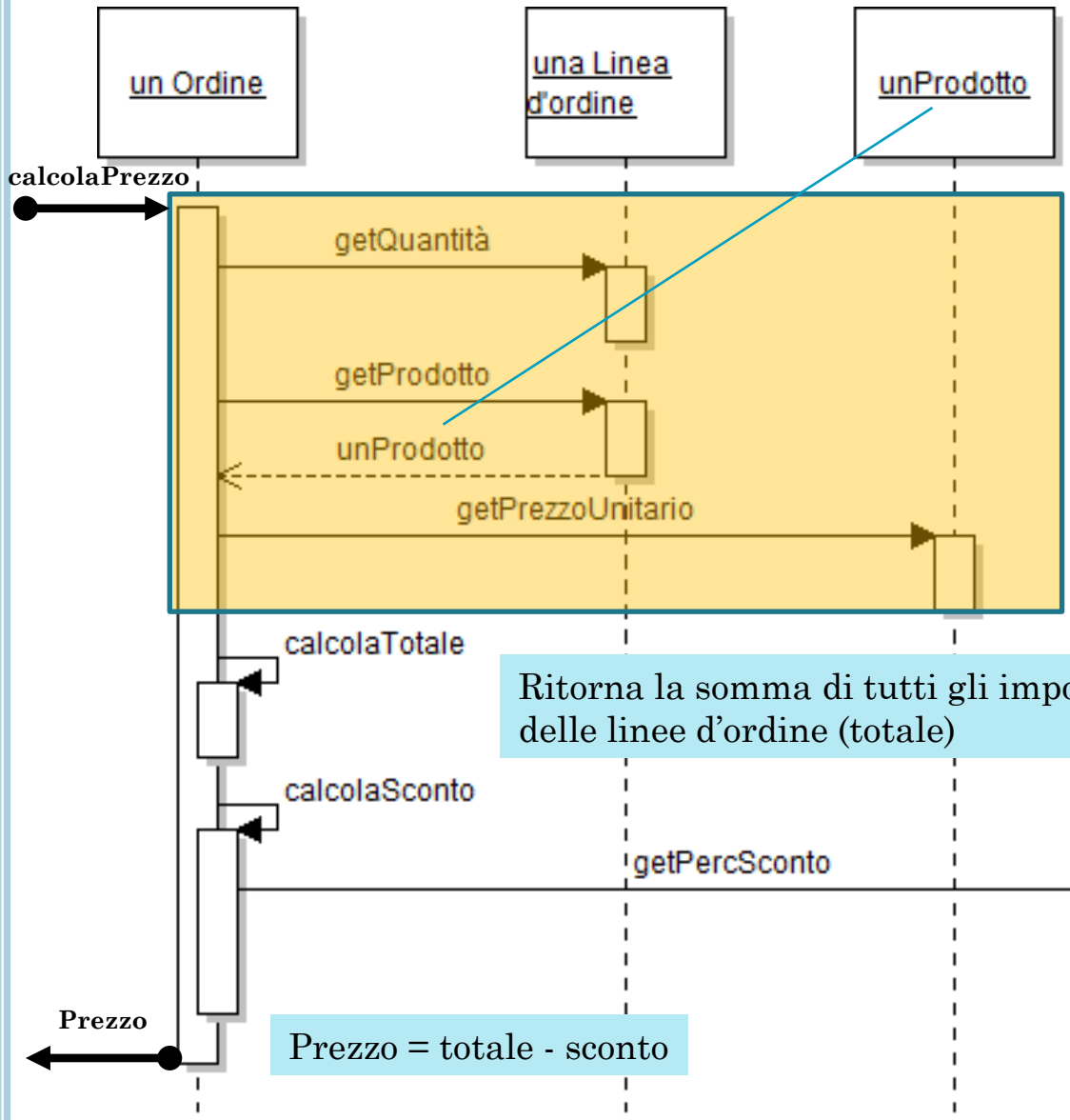
Data dell'ordine 18/10/2011

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**





SEQUENCE DIAGRAM: PRIMA SOLUZIONE



Fowler

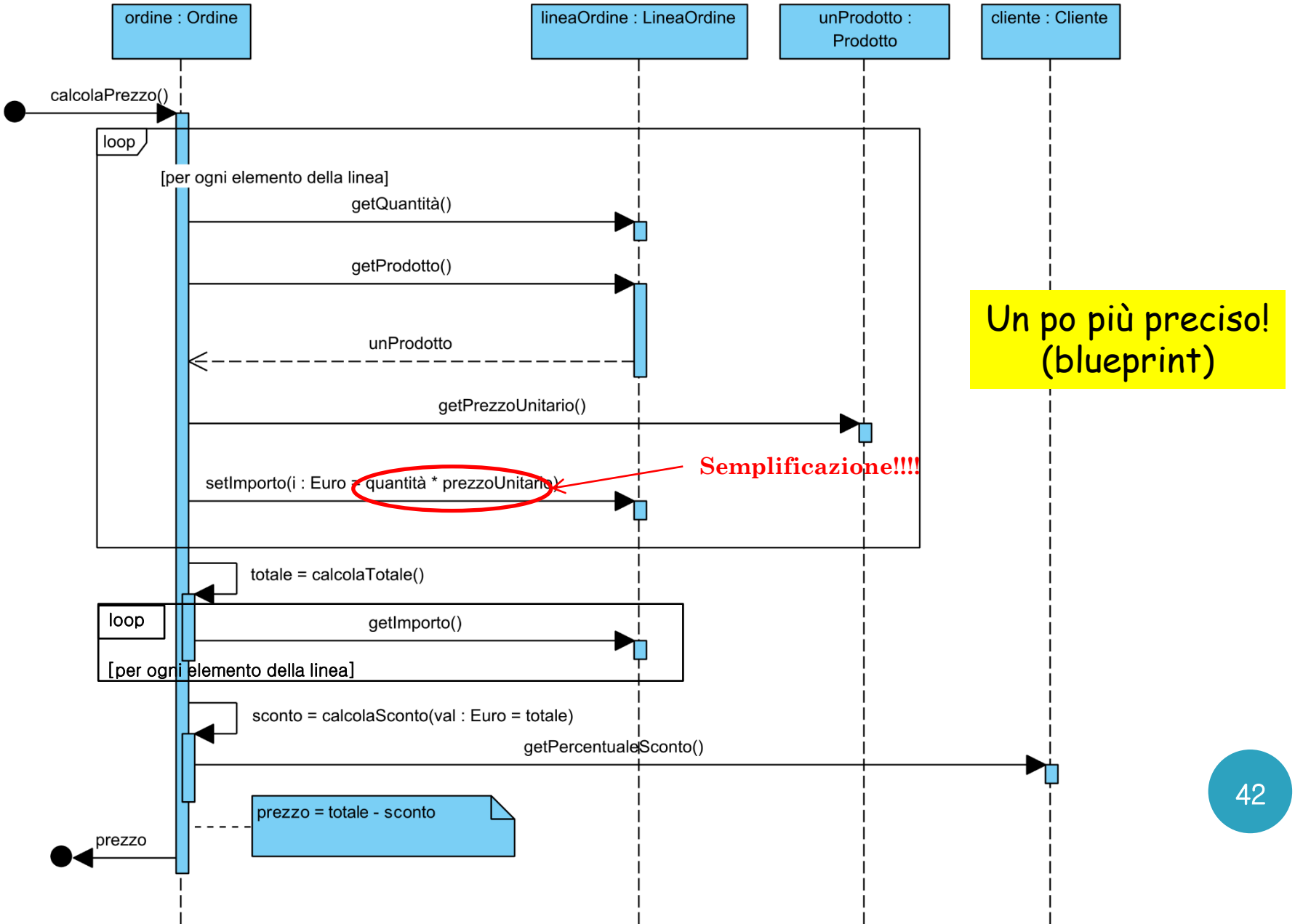
Manca il fatto che le tre operazioni sono chiamate per ogni linea d'ordine!

Ritorna la somma di tutti gli importi delle linee d'ordine (totale)

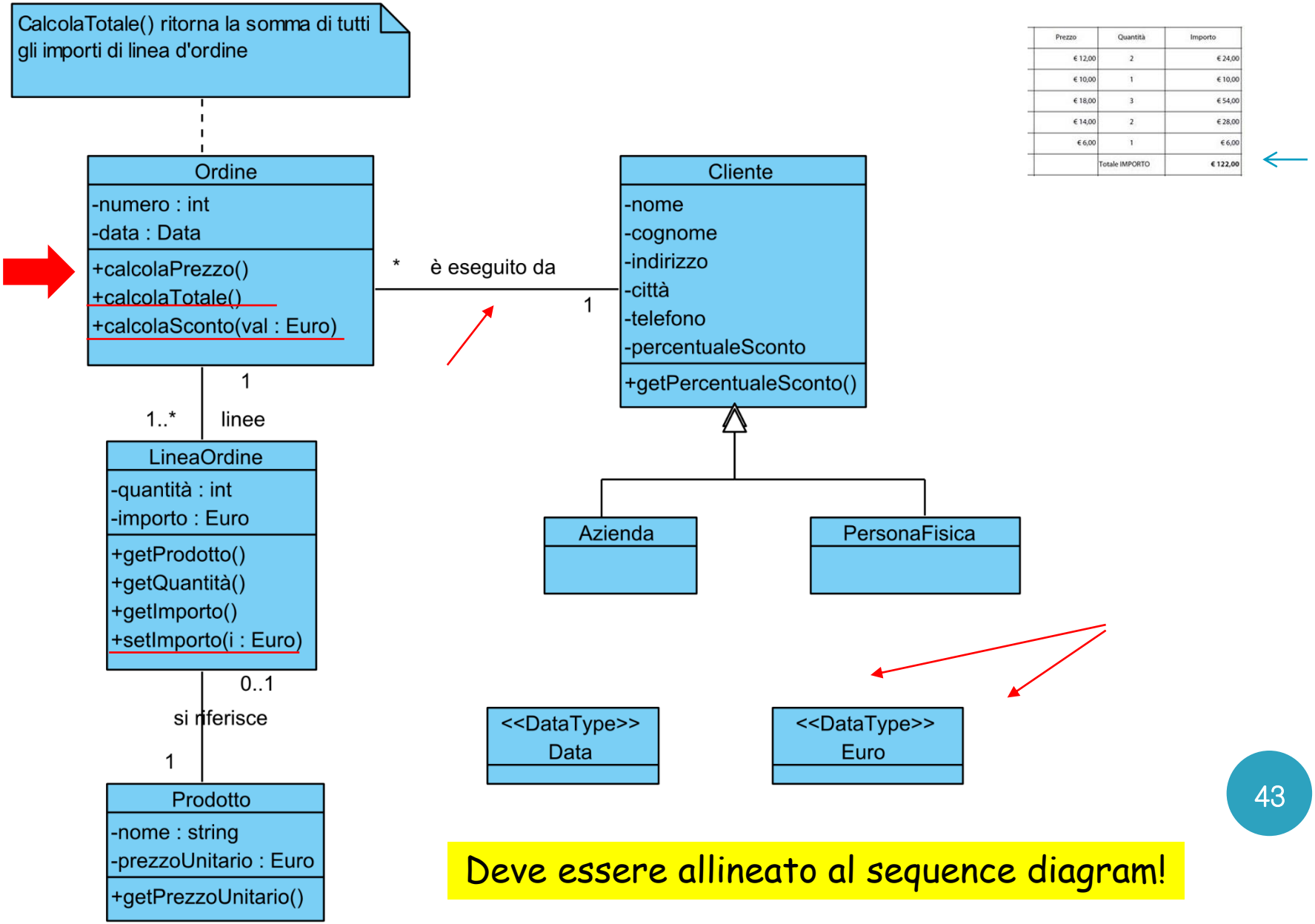
Prodotto	Prezzo	Quantità	Importo
Prodotto A	€ 12,00	2	€ 24,00
Prodotto B	€ 10,00	1	€ 10,00
Prodotto C	€ 18,00	3	€ 54,00
Prodotto D	€ 14,00	2	€ 28,00
Prodotto E	€ 6,00	1	€ 6,00
		Totale IMPORTO	€ 122,00

Prezzo = totale - sconto

SEQUENCE DIAGRAM: PRIMA SOLUZIONE



Class diagram: prima soluzione



Deve essere allineato al sequence diagram!

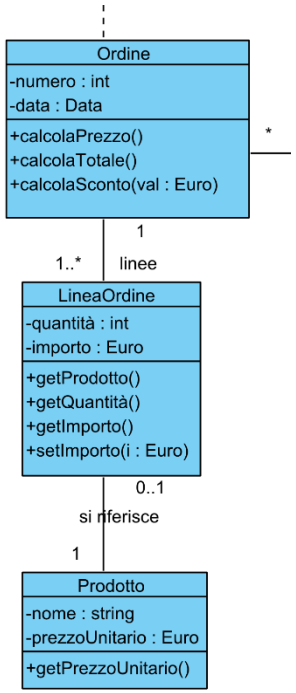
PSEUDOCODICE: PRIMA SOLUZIONE

+, * e - si assumono per Euro/int

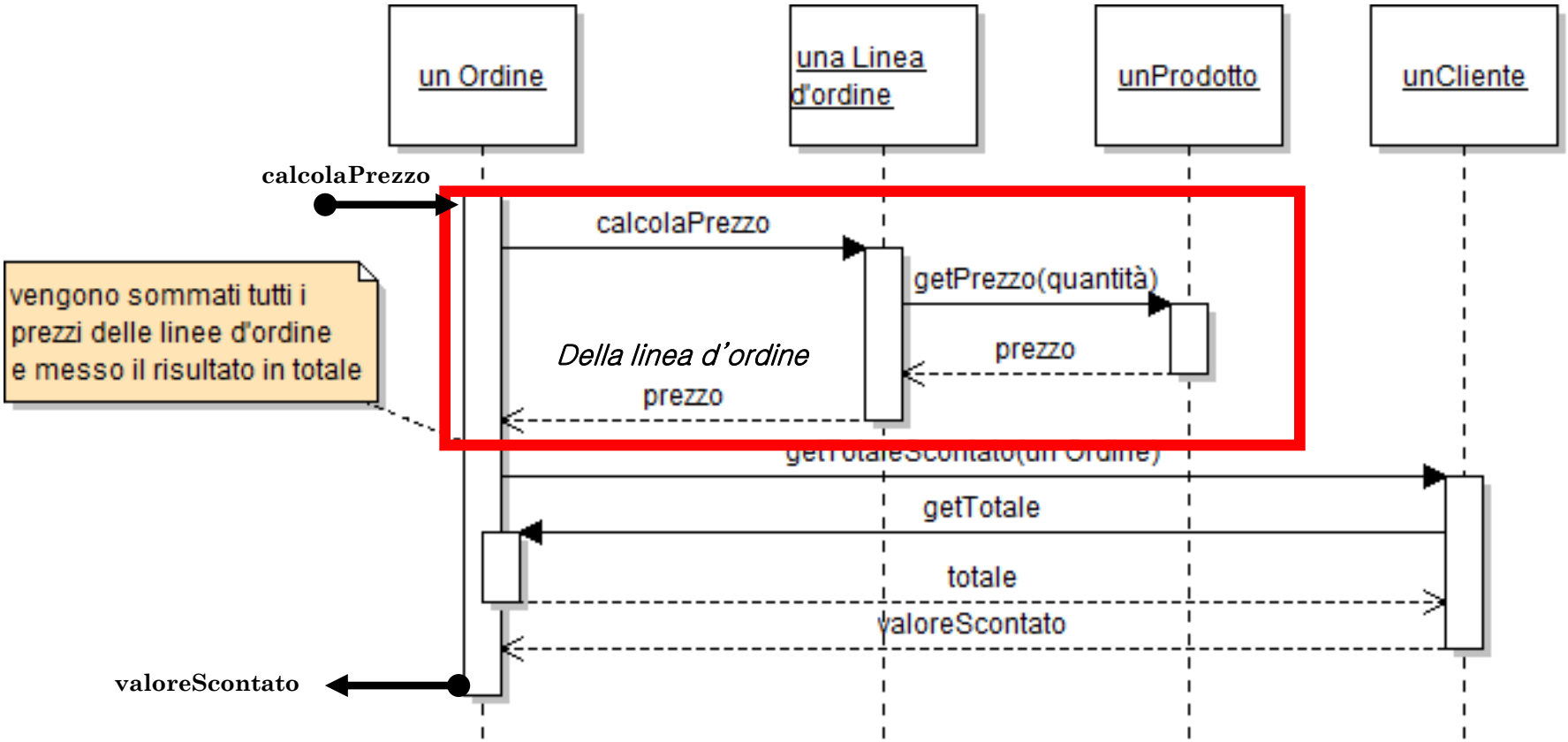
```
Public class Ordine {
  private HashSet<Linea> linee;
  private Cliente cliente;
  ...
  public Euro calcolaPrezzo() {
     int quantità;
     Prodotto p;
     Euro prezzoUnitario, sconto, totale;
     For (Linea linea: linee) {
        quantità = linea.getQuantità();
        p = linea.getProdotto();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

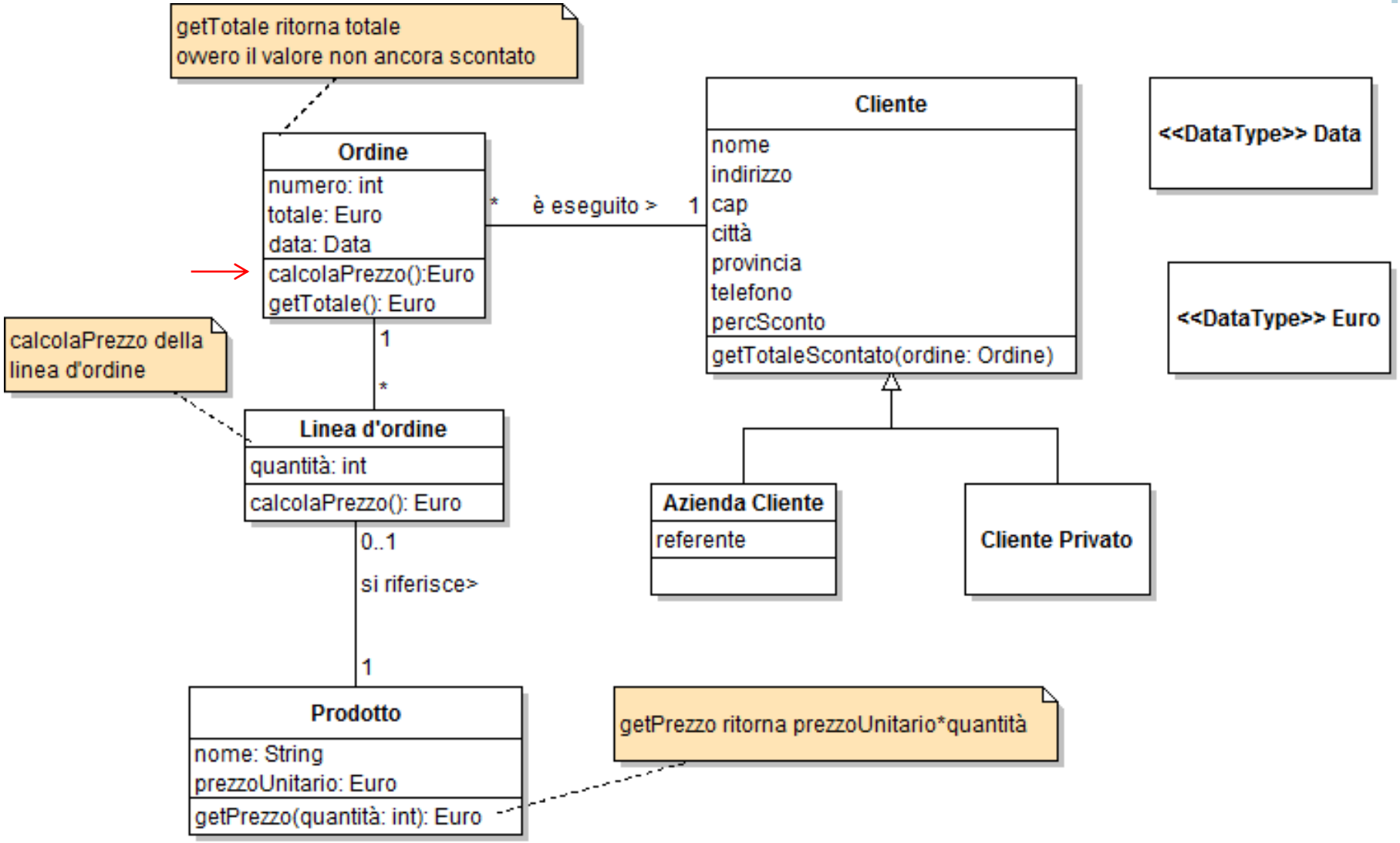
```
private Euro calcolaSconto(Euro t) {
      int perc = cliente.getPercSconto();
      return t*perc/100
}
```



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;
  }
}
```

```
Public class Linea {
  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;
  }
}
```

Confronto

- Nella prima soluzione un partecipante (**oggetto ordine**) svolge tutta l'elaborazione mentre gli altri forniscono solo i dati
  - **Controllo centralizzato**

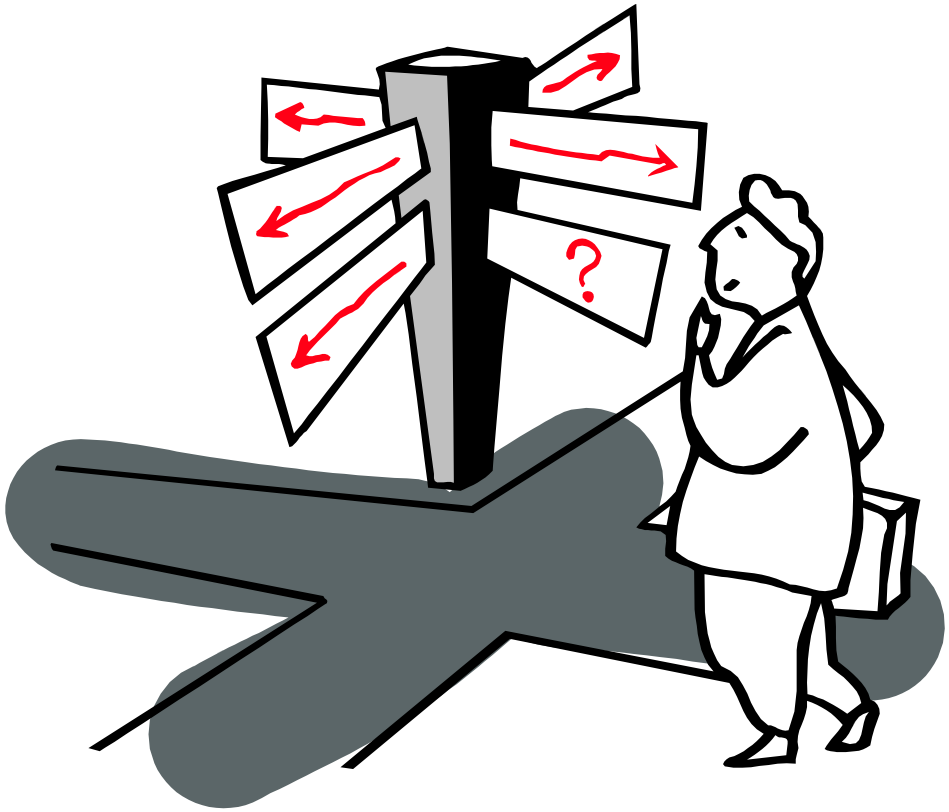
- Nella seconda soluzione invece ogni partecipante contribuisce in parte all'elaborazione dell'algoritmo
  - **Controllo distribuito**

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?