08 - Modellazione statica e dinamica con UML

Dai Requisiti alla Progettazione

Nella fase dei requisiti, l'attenzione è sul "fare la cosa giusta" (capire gli obiettivi principali, regole, vincoli).
Nella progettazione, l'enfasi è sul "fare la cosa bene" (progettare abilmente una soluzione che soddisfi i requisiti).

> È naturale scoprire e modificare alcuni requisiti durante il lavoro di progettazione e implementazione, soprattutto nelle iterazioni iniziali; programmazione precoce, test e demo aiutano a provocare questi cambiamenti inevitabili (scopo dello sviluppo iterativo).

Verso la Progettazione ad Oggetti

Come progettare a oggetti?

- **Codifica**: Progettare mentre si codifica.
- **Disegno, poi codifica**: Disegnare alcuni diagrammi UML, poi passare alla codifica (approccio raccomandato, con "disegno leggero"). Il costo aggiuntivo del disegno dovrebbe ripagare lo sforzo.
- **Solo disegno**: Lo strumento genera tutto dai diagrammi.

La **modellazione agile** riduce il costo aggiuntivo del disegno e modella per comprendere e comunicare, non per documentare.
Le pratiche includono:

- modellare insieme agli altri (modellazione in gruppo)
- creare diversi modelli in parallelo (dinamici e statici).

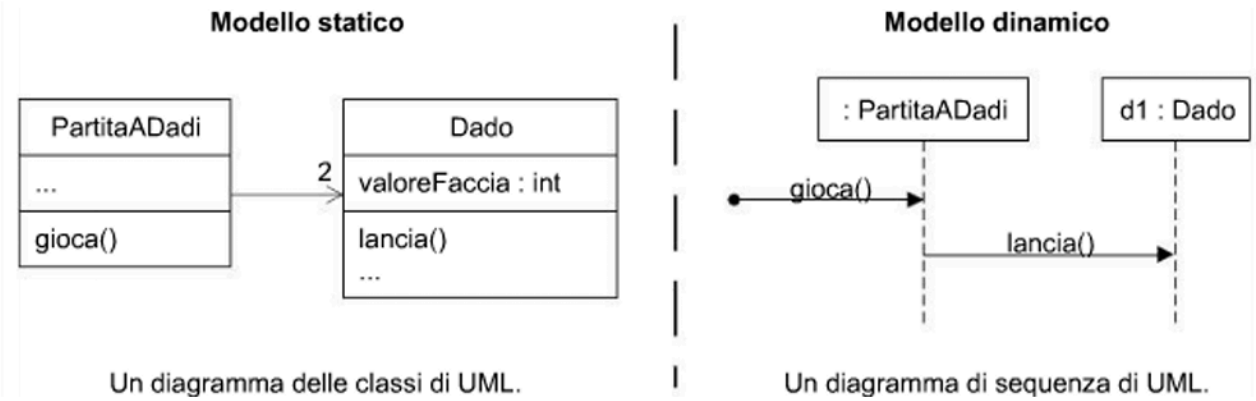
Modellazione Statica e Dinamica

Ci sono due tipi di modelli per gli oggetti:

- **Modelli dinamici** (es. diagrammi di interazione UML): Rappresentano il comportamento del sistema, la collaborazione tra oggetti software per realizzare scenari di casi d'uso, i metodi di classi software. La modellazione a oggetti dinamica più comune è con i diagrammi di sequenza di UML.
- **Modelli statici** (es. diagrammi di classe UML): Servono per definire i package, i nomi delle classi, gli attributi, le firme di operazioni. La modellazione a oggetti statica più comune è con i diagrammi delle classi di UML.

I modelli dinamici e statici sono tra loro relazionati ed è per questa ragione che si consiglia di crearli in parallelo.

I messaggi nel diagramma di sequenza indicano operazioni nelle classi che ricevono il messaggio del diagramma delle classi, e le linee di vita nel diagramma di sequenza rappresentano oggetti di classi del diagramma delle classi.



La maggior parte del lavoro di progettazione difficile e utile avviene mentre si disegnano i diagrammi di interazione (vista dinamica). Durante la modellazione a oggetti dinamica, si pensa in modo dettagliato e preciso a quali oggetti devono esistere e come collaborano tramite messaggi e metodi.
Durante questa fase si applicano la progettazione guidata dalle responsabilità e i principi **GRASP**.

> Quali sono le responsabilità dell'oggetto?
> Con chi collabora l'oggetto?
> Quali design pattern devono essere applicati?

La progettazione a oggetti richiede conoscenza di **principi di assegnazione di responsabilità** e **design pattern**.

Diagrammi di Interazione

UML comprende i **diagrammi di interazione** per illustrare come gli oggetti interagiscono tramite lo scambio di messaggi. Sono usati per la **modellazione dinamica** degli oggetti.

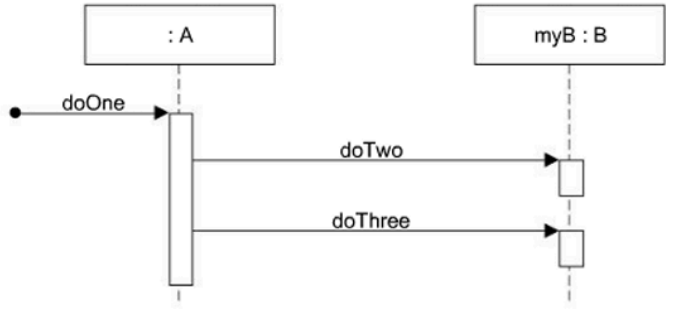
Un'**interazione** è una specifica di come alcuni oggetti si scambiano messaggi nel tempo per eseguire un compito. È una generalizzazione dei diagrammi di **sequenza** e di **comunicazione**.

- Un'**interazione** è motivata dalla necessità di eseguire un determinato **compito**, rappresentato da un **messaggio** che dà inizio all'interazione (messaggio trovato).
- Il **messaggio** è inviato a un oggetto designato come **responsabile**, che collabora/interagisce con altri oggetti (**partecipanti**) per svolgere il compito.

- Ogni **partecipante** svolge un proprio **ruolo** nella **collaborazione** tramite **scambio di messaggi**, e ogni messaggio è una richiesta di un oggetto a un altro per eseguire un'**operazione**.

Diagrammi di Sequenza

I **diagrammi di sequenza** mostrano le interazioni in un formato a "steccato" con gli oggetti partecipanti in alto.



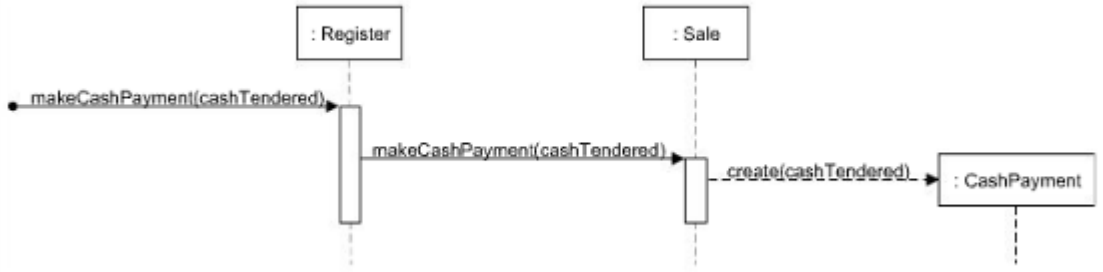

```
public class A {
     private B myB = new B();

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

Vantaggi: Mostrano chiaramente la sequenza temporale dei messaggi.

Svantaggi: Costringono a estendersi verso destra con nuovi oggetti.

Esempio:

- il messaggio *makeCashPayment* viene inviato a un'istanza di *Register*. Il mittente non è identificato.
- L'istanza di *Register* invia a un'istanza di *Sale*
- L'istanza di *Sale* crea un'istanza di *CashPayment*.

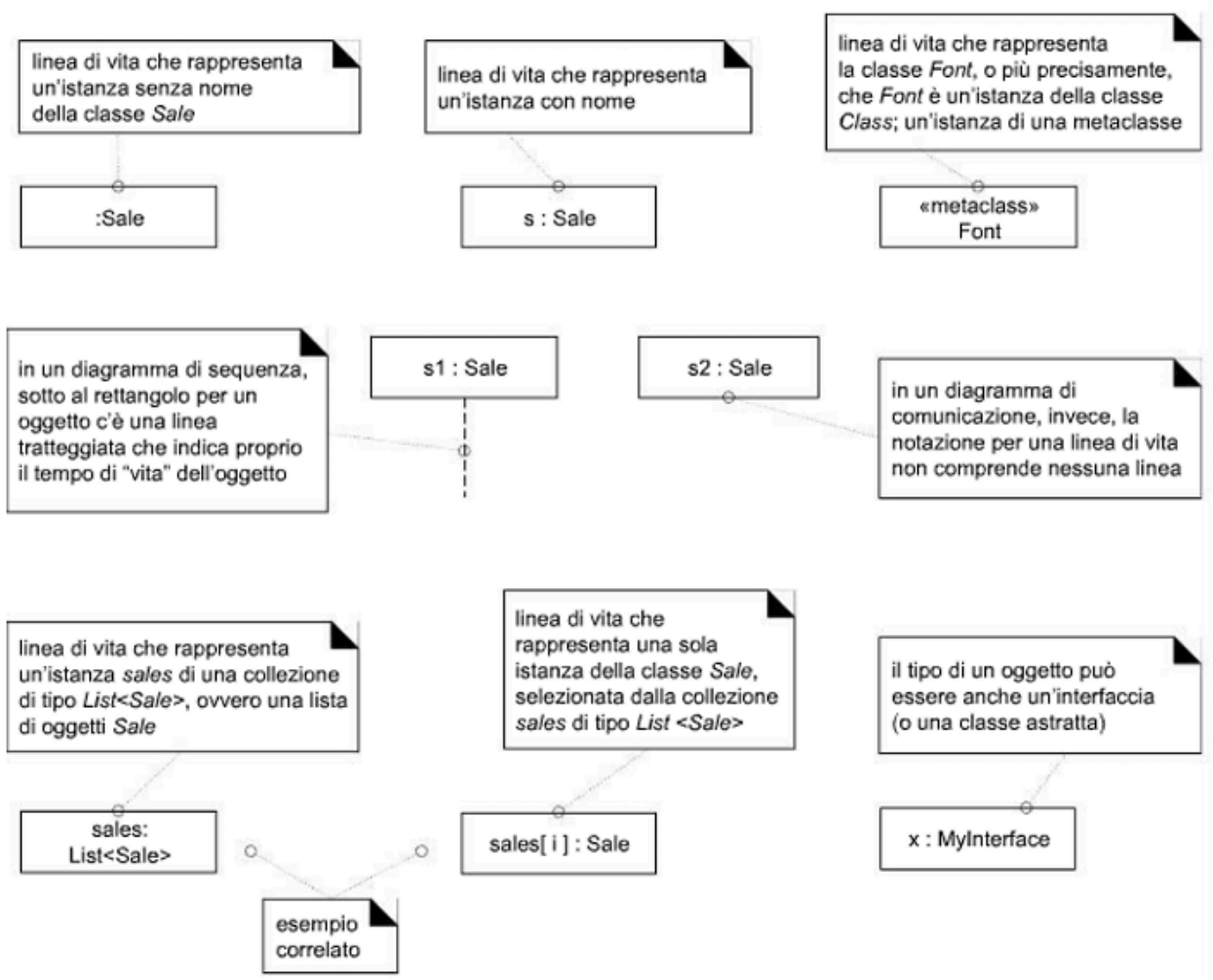



```
public class Sale {
     private CashPayment payment;

     public void makeCashPayment( Money cashTendered )
          payment = new CashPayment( cashTendered );
          ...
     }
     ...
}
```

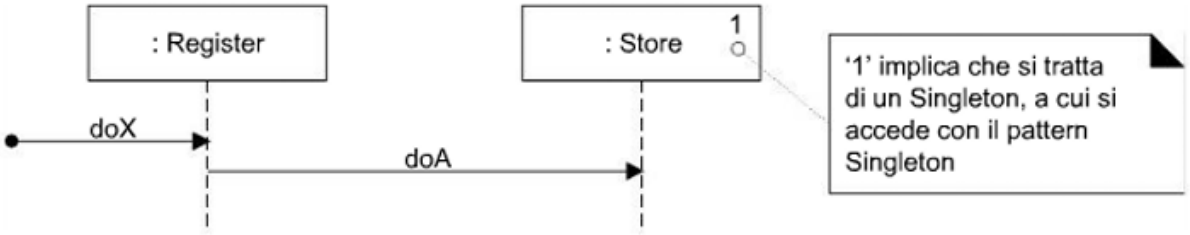
In **UP**, un **Design Sequence Diagram** (**DSD**) è un diagramma di sequenza usato dal punto di vista software o di progetto. L'insieme di tutti i DSD fa parte del **Modello di Progetto**.

Notazione per Diagrammi di Sequenza:

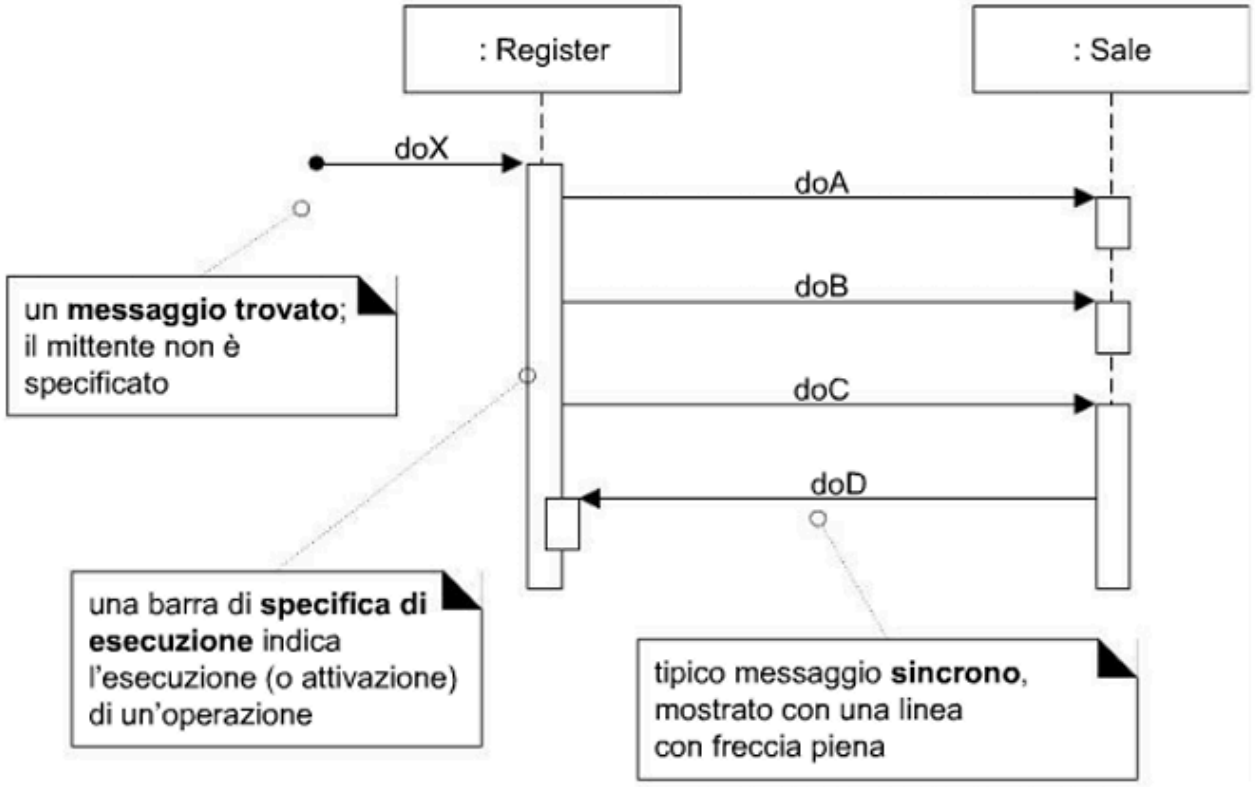
- **Partecipanti**:
  Rettangoli chiamati **linee di vita** (*lifeline*).



- **Singleton**:
  In un diagramma di interazione, un oggetto "singleton" è contrassegnato da un '`1`' nell'angolo superiore destro della linea di vita.

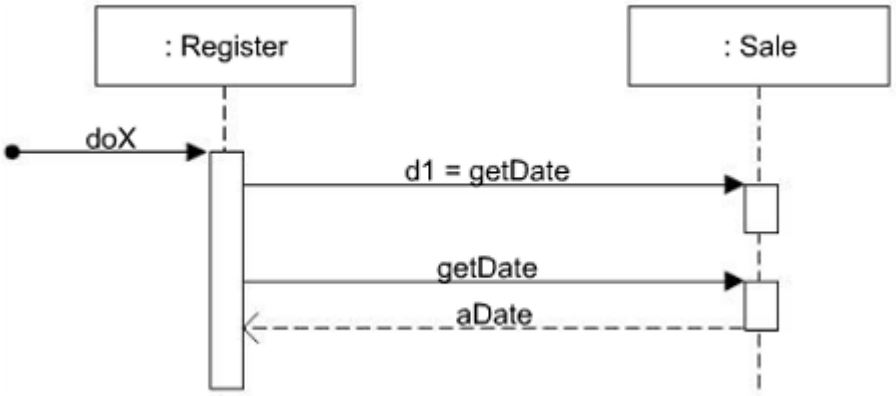


- Una **linea di vita** include un rettangolo e una linea verticale che si estende sotto di esso.
- I **messaggi** sono linee continue con frecce piene (**sincroni**) o sottili (**asincroni**). Il messaggio iniziale è il "***messaggio trovato***".
- Una barra di **specifica di esecuzione** (**attivazione**) mostra l'esecuzione di un'operazione.

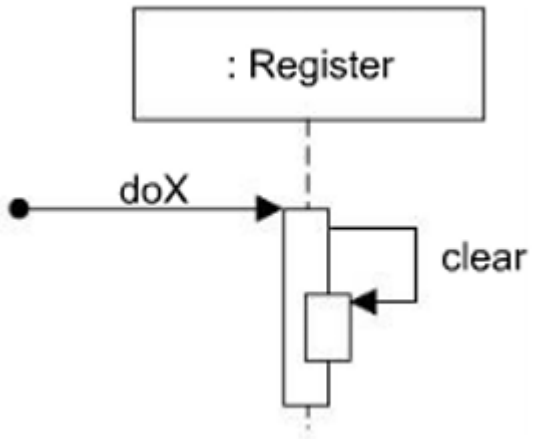


Ci sono due modi per mostrare il risultato di ritorno:

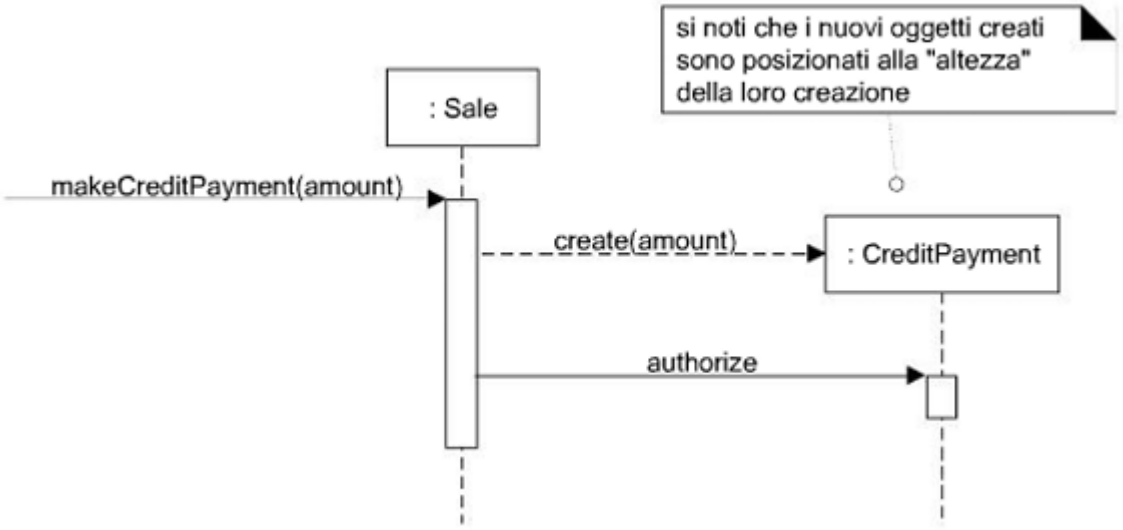
- sintassi `returnVar = message(parametri)`
- una linea di messaggio di risposta alla fine della barra di attivazione.



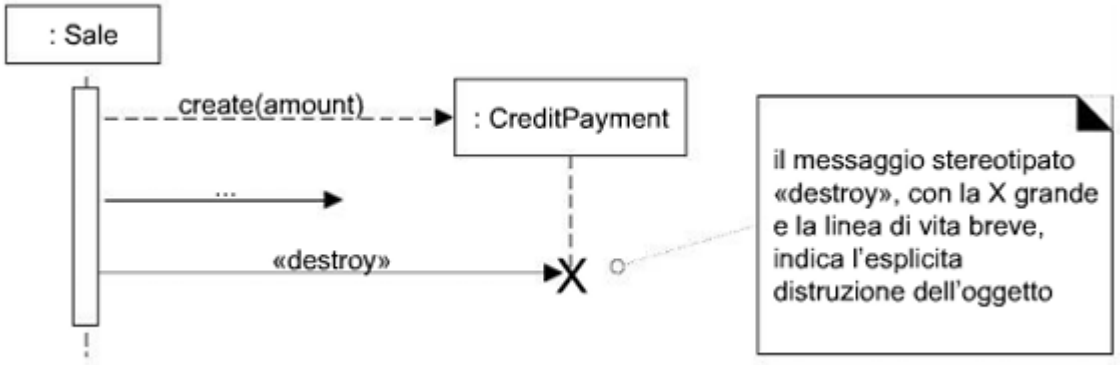
Self o **this**:
Messaggio inviato da un oggetto a se stesso con barra di esecuzione annidata.



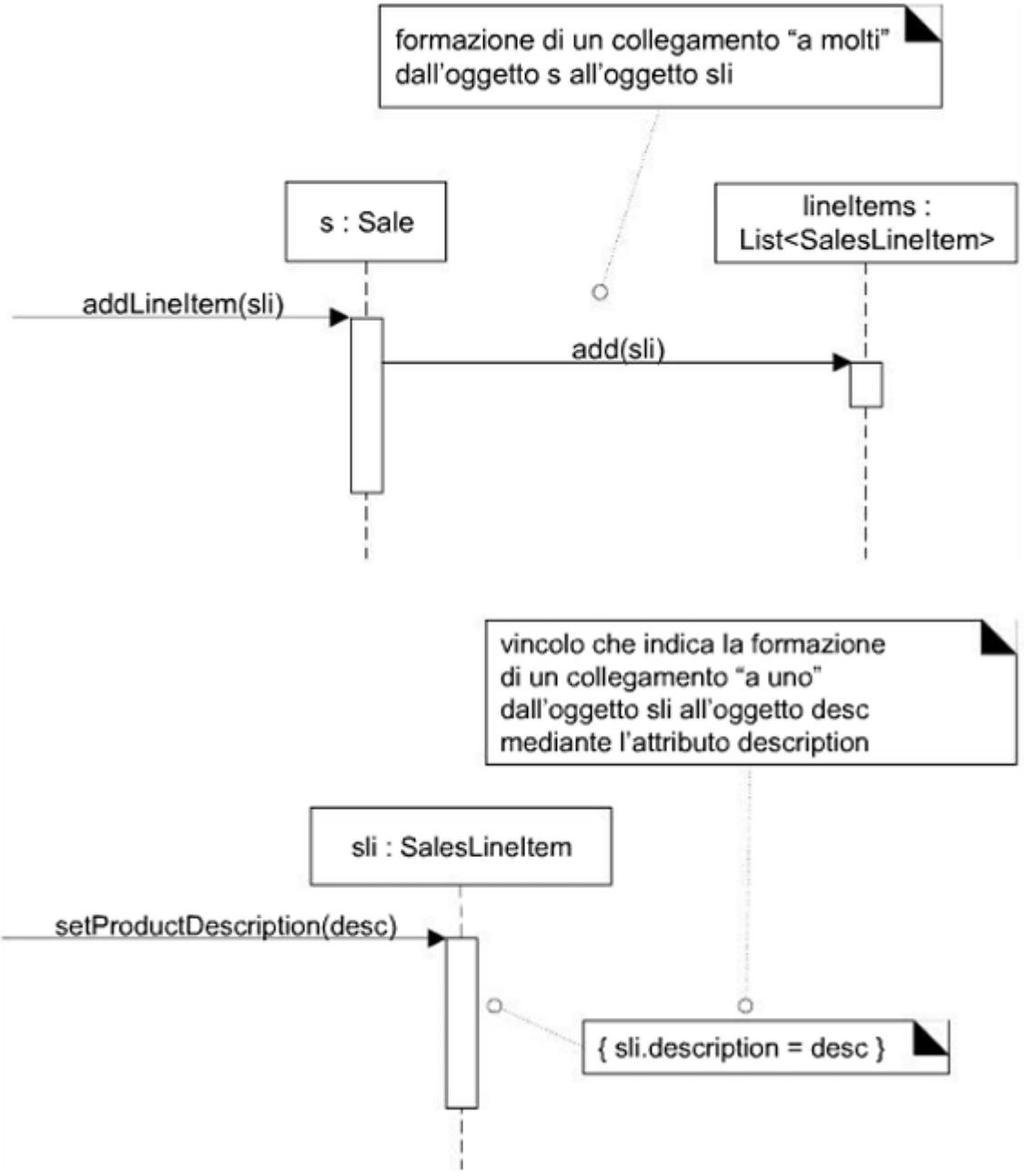
Creazione di istanze:
I nuovi oggetti creati sono posizionati all'"altezza" della loro creazione.



Distruzione di oggetti:

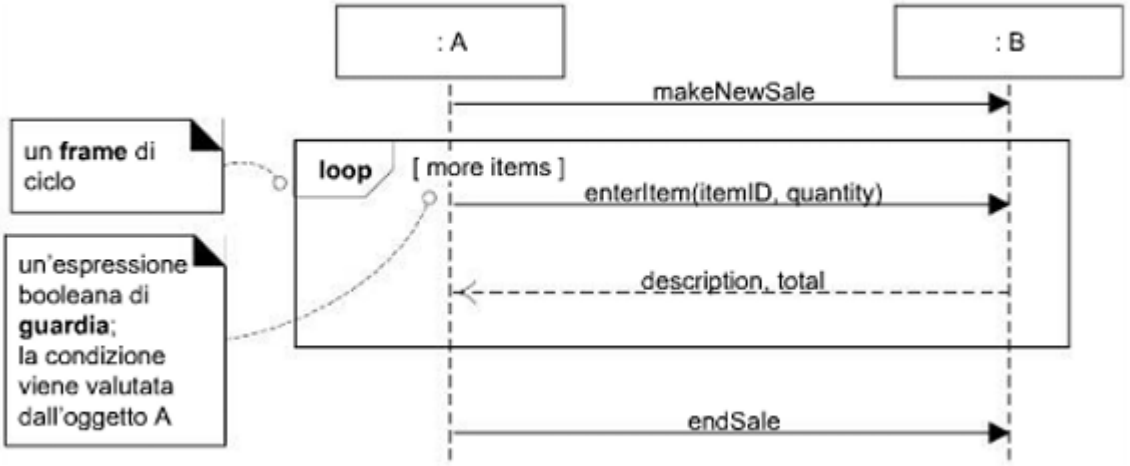


Formazione di collegamenti:
Per associazioni "a molti" o "a uno".
Il vincolo è una post-condizione dell'operazione che deve risultare vera al termine della sua esecuzione.



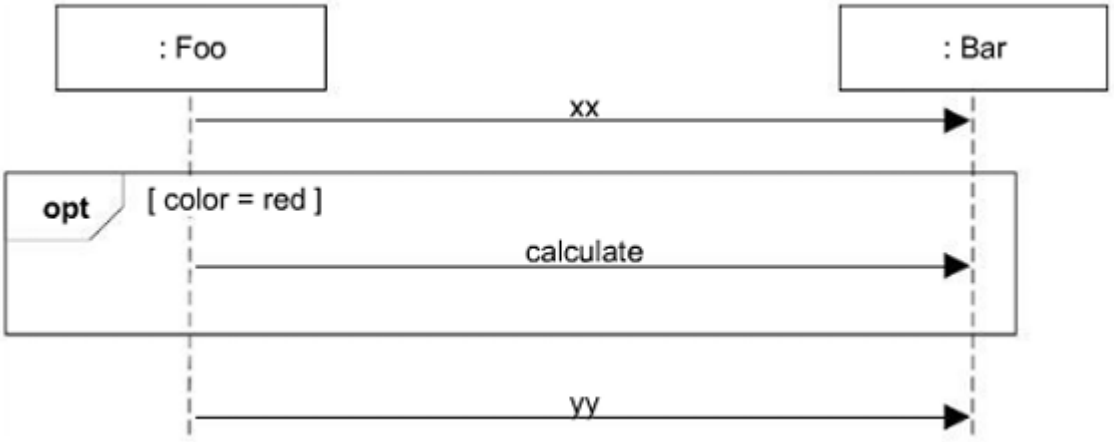
Frame:
Supporto per istruzioni di controllo condizionali e di ciclo.

Sono regioni con un operatore (etichetta) e una guardia (condizione booleana).

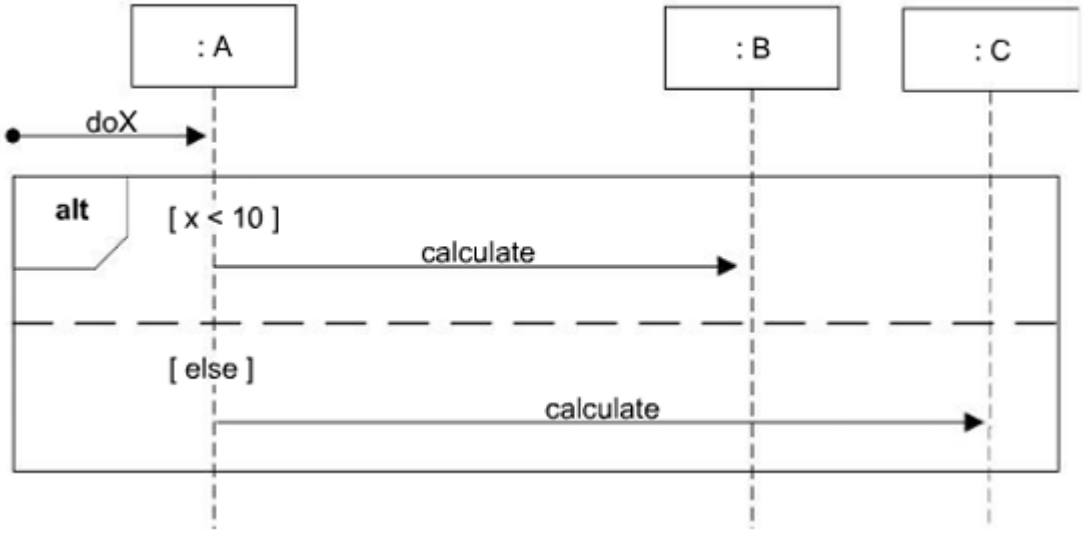


Messaggi condizionali:

- Frame *opt* è posizionato attorno a uno o più messaggi

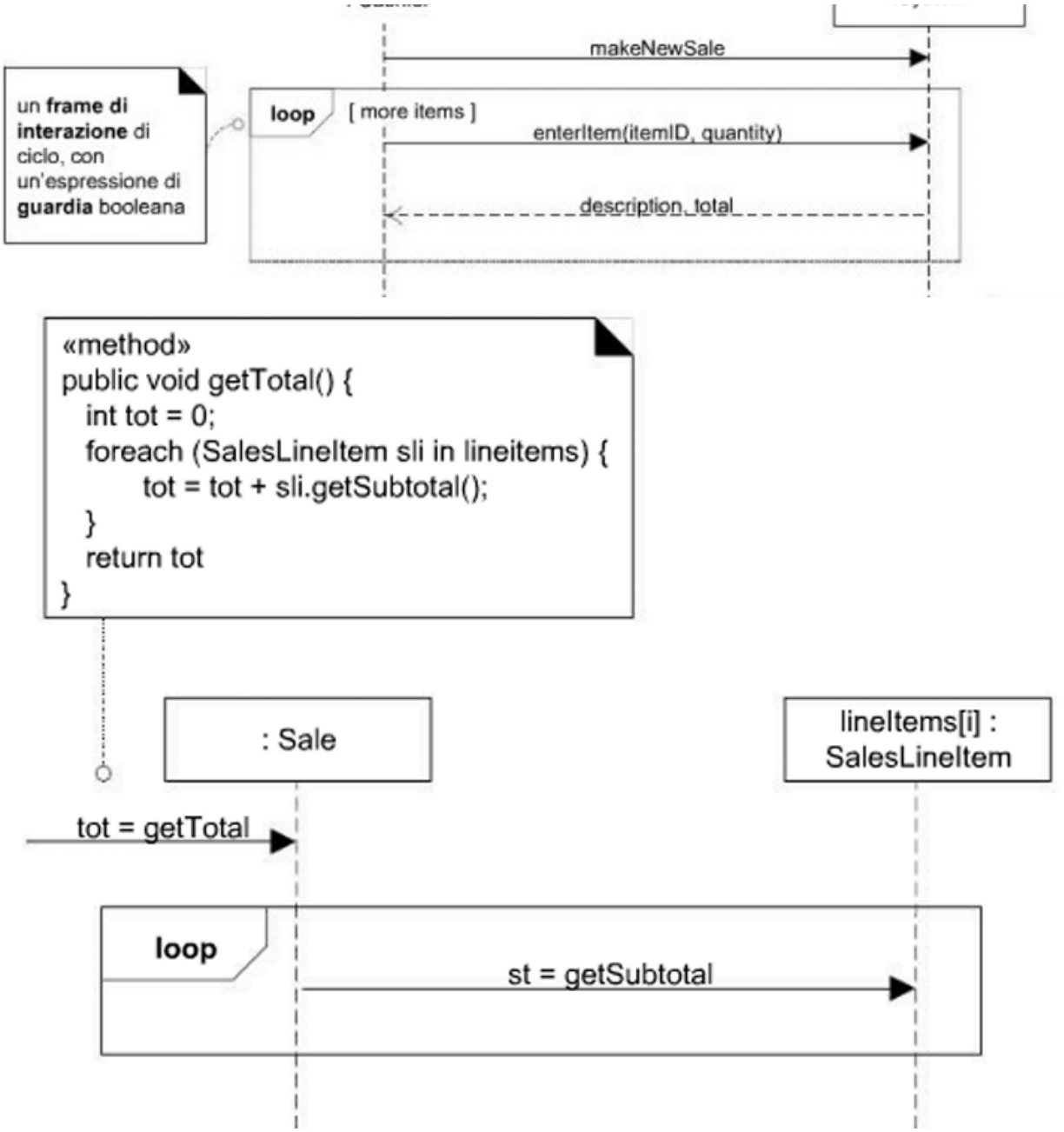


- Frame *alt* è posizionato attorno alle alternative mutualmente esclusive



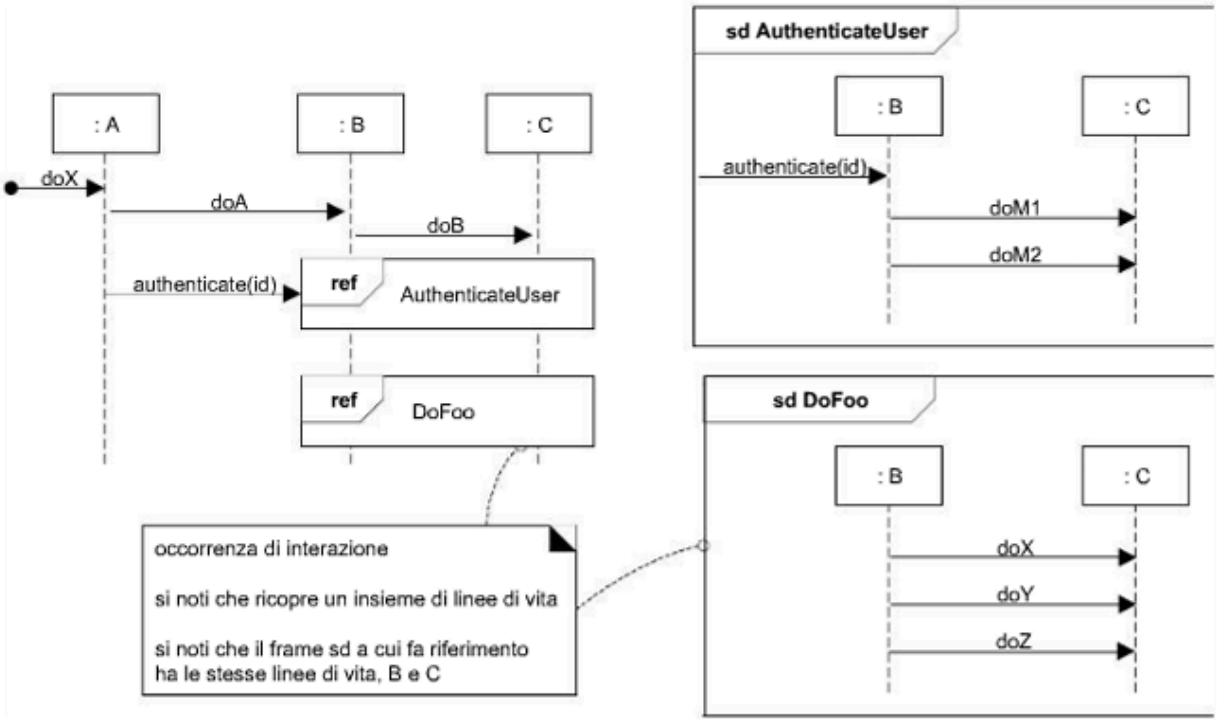
Iterazioni:

- Frame *loop* è posizionato attorno a uno o più messaggi da iterare



I frame possono essere annidati.

Un'occorrenza di interazione (o uso di interazione) è un riferimento a un'interazione all'interno di un'altra.



Invocare metodi statici:
L'oggetto ricevente è una classe o un'istanza di una meta-classe.

Chiamate sincrone e asincrone:
Notare la distinzione sottile nelle frecce.

Un *oggetto attivo* è eseguito nel proprio thread di esecuzione e lo controlla.

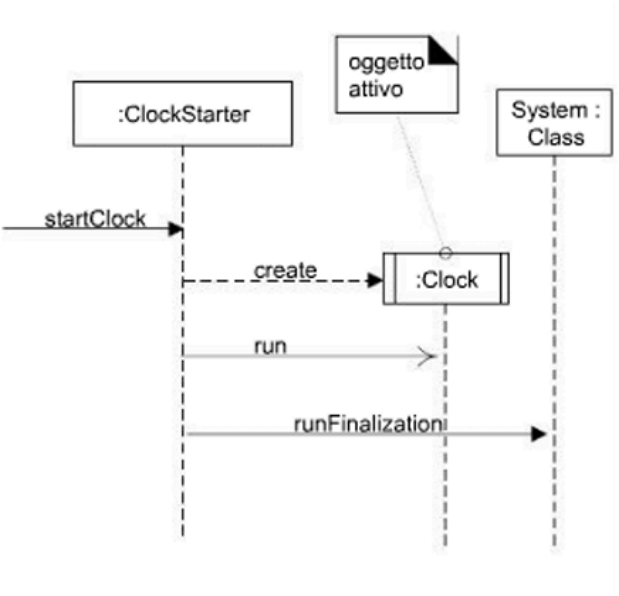
una freccia non piena indica una **chiamata asincrona** (diversamente da una freccia piena, che invece rappresenta una più comune chiamata sincrona)

in Java, per esempio, una chiamata asincrona può avvenire come segue:

```
// Clock implementa l'interfaccia Runnable
Thread t = new Thread( new Clock() );
t.start();
```

la chiamata di *start* implica poi una chiamata asincrona del metodo *run* dell'oggetto *Runnable* (*Clock*)

per semplificare il diagramma, l'oggetto *Thread* e il messaggio *start* si possono evitare (sono un "overhead" standard); al contrario, la chiamata asincrona è implicata dai dettagli essenziali della creazione di *Clock* e dal messaggio *run*



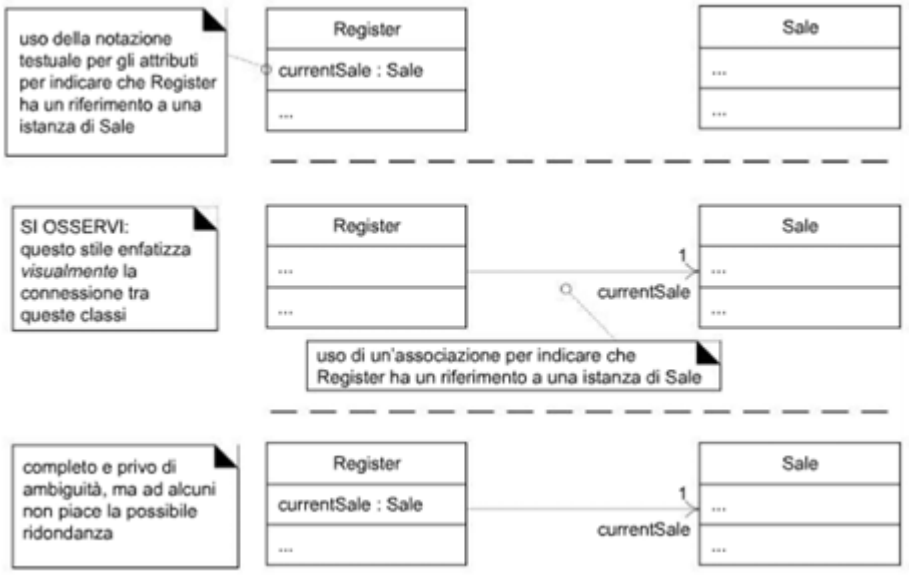
Diagrammi delle Classi

UML include i **diagrammi delle classi** per illustrare classi, interfacce e relative associazioni.
Sono usati per la **modellazione statica degli oggetti**.
Sono stati usati concettualmente per visualizzare un modello di dominio.

Un **Design Class Diagram** (**DCD**) è un diagramma delle classi usato da un punto di vista software o di progetto. In UP, tutti i DCD fanno parte del **Modello di Progetto**.

Notazione comune per DCD

- Notazione testuale per attributi
- Notazione con linea di associazione



Non si consiglia di usarle entrambe insieme.
Se non viene indicata alcuna visibilità, si ipotizza che gli attributi siano privati.

Freccia di navigabilità

Dalla classe sorgente alla classe destinazione, indica che un oggetto della sorgente ha un attributo del tipo della destinazione.

Molteplicità

All'estremità vicina alla destinazione.

Nome di ruolo

- Solo all'estremità vicina alla destinazione per indicare il nome dell'attributo.
- Nessun nome per l'associazione.

Operazioni e metodi

- Un'operazione è una dichiarazione di un metodo:
  `visibility name (parameter-list) : return-type { property-string }`
- Di default, le operazioni hanno visibilità pubblica.
- I diagrammi di classe indicano le operazioni (signature), mentre i diagrammi di interazione modellano i metodi come sequenze di messaggi.

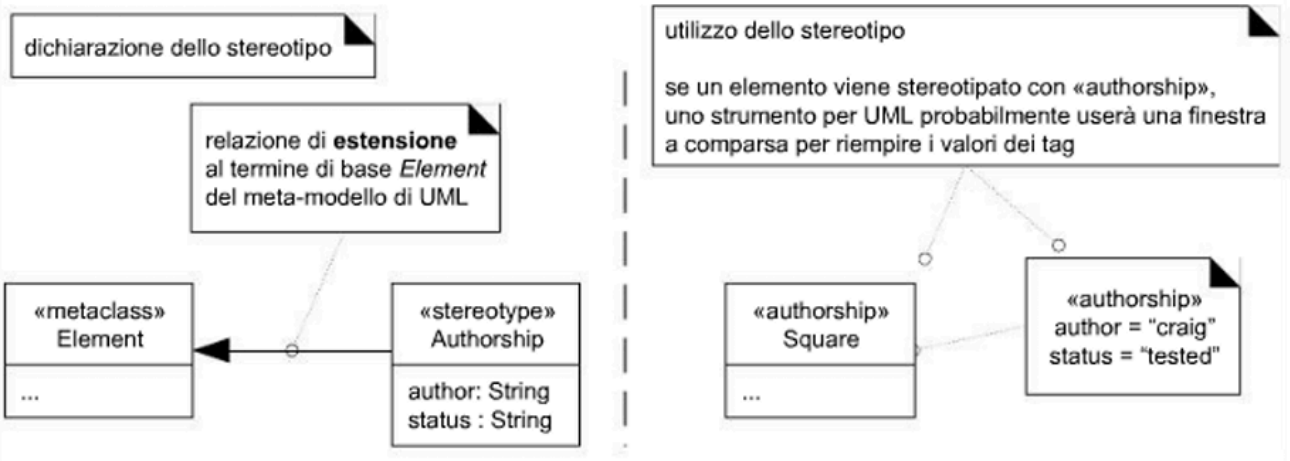
Parole chiave

Decoratori testuali per classificare un elemento.

Parola Chiave	Significato	Esempio d'uso
`«actor»`	il classificatore è un attore	nei diagrammi delle clasi, sopra al nome di un classificatore
`«interface»`	il classificatore è un'interfaccia	nei diagrammi delle classi, sopra al nome di un classificatore
`{abstract}`	l'elemento è astratto; non può essere istanziato nome di un'operazione	nei diagrammi delle classi, dopo il nome di un classificatore
`{ordered}`	un insieme di oggetti ha un ordine predefinito	nei diagrammi delle classi, a un'estremità di associazione

Stereotipi

Rappresentano un raffinamento di un concetto di modellazione esistente, definito in un profilo UML (un profilo e una collezione di stereotipi).

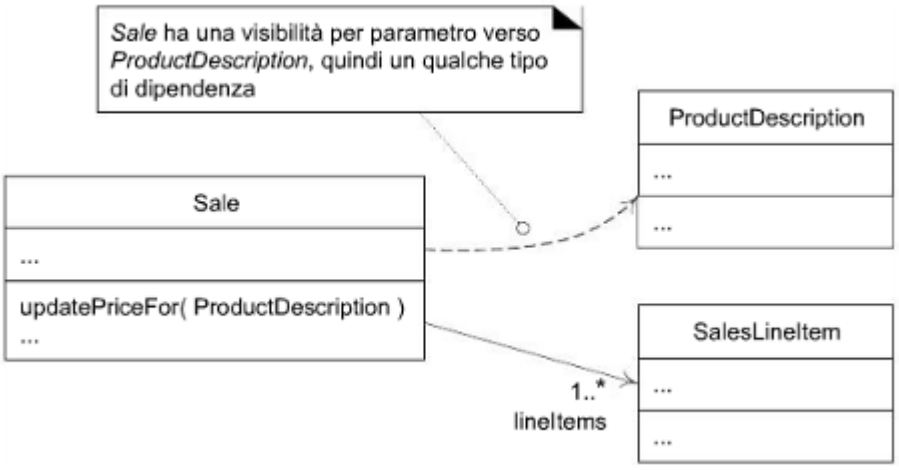


Generalizzazione

- **Relazione tassonomica** tra un classificatore più generale e uno più specifico.
- **Implica ereditarietà** nei linguaggi OO.
- Si usa il tag `{abstract}` per le classi astratte.

Dipendenze

- Le **linee di dipendenza** sono comuni nei diagrammi di classe e package.
- Una dipendenza indica che un elemento *cliente* è a conoscenza di un *fornitore* e un cambiamento nel fornitore potrebbe influire sul cliente (**accoppiamento**).



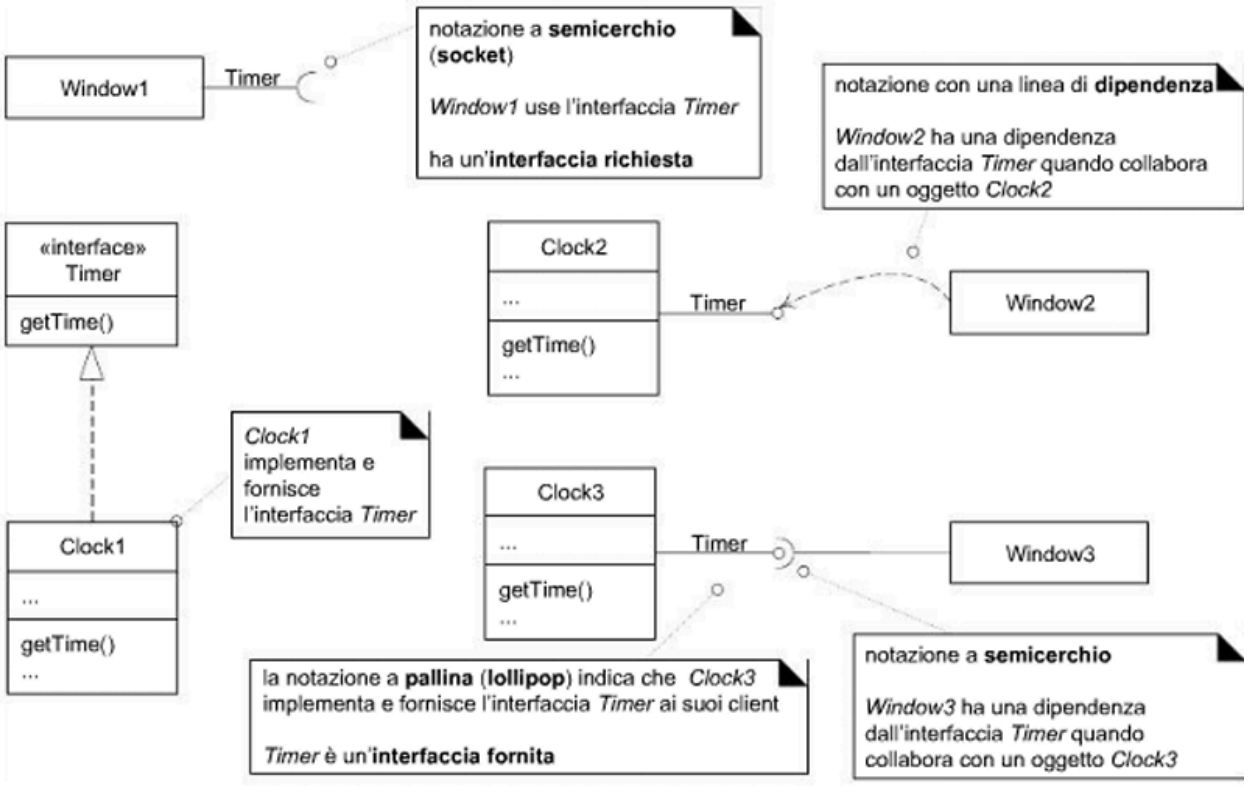
Esistono molte tipologie di dipendenze:

- Avere un attributo del tipo del fornitore,
- inviare un messaggio a un fornitore (visibilità data da attributo, parametro, variabile locale/globale, visibilità di classe per metodi statici),
- ricevere un parametro del tipo del fornitore,
- fornitore è superclasse o interfaccia implementata.

Interfacce:

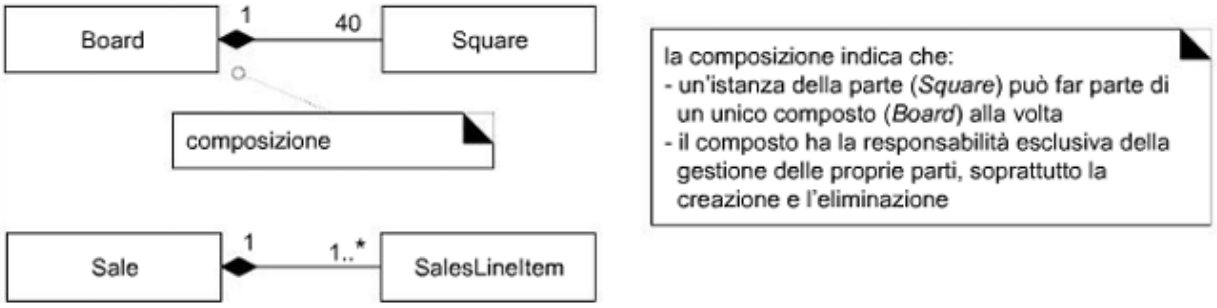
- L'implementazione di un'interfaccia è chiamata **realizzazione di interfaccia**.

- *Notazione a pallina* (**lollipop**):
  Classe X implementa (fornisce) interfaccia Y.
- *Notazione a semicerchio* (**socket**):
  Classe X richiede (usa) interfaccia Y.



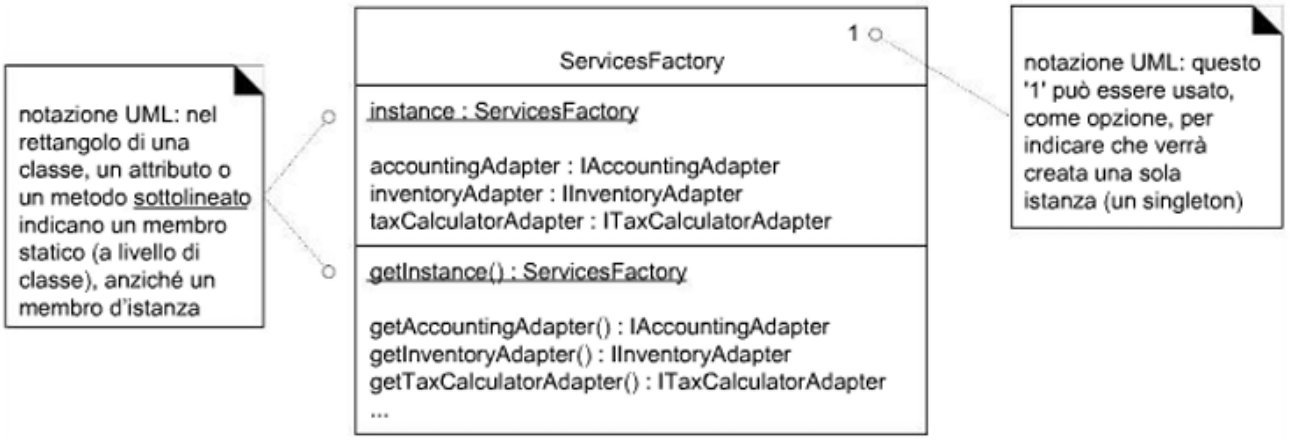
Composizione (punto di vista software)

- Gli oggetti B non possono esistere indipendentemente da A;
- A è responsabile della creazione e distruzione dei suoi oggetti B.



Singleton

Esiste **una sola istanza** di una classe.



Template

Molti linguaggi supportano **tipi a template** (**tipi parametrizzati**, **generici**).