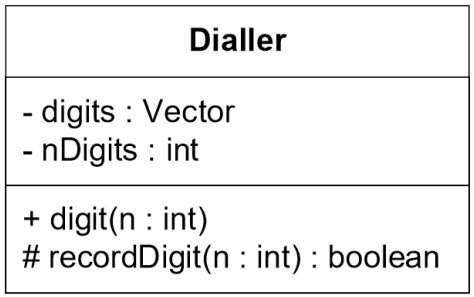
Modellare la struttura

Diagramma delle Classi con prospettiva software

- Prospettiva software
  - Gli elementi di un modello corrispondono agli elementi di un sistema software orientato agli oggetti
  - È indipendente dal linguaggio di programmazione: Java, C++, C#, Python, …
- Il diagramma delle classi con prospettiva software descrive il tipo degli oggetti che fanno parte di un sistema e le varie tipologie di relazioni statiche fra di essi

Classe


```
public class Dialler
{
}
```




```
public class Dialler
{
  private Vector digits;
  int nDigits;
  public void digit(int n);
  protected boolean recordDigit(int n);
}
```

Attributi

- Rappresentano le **proprietà strutturali** degli oggetti di una classe
- In un linguaggio OO corrispondono al nome di una **variabile di istanza** della classe

Nome classe
Nome attributo Nome attributo: tipo dati Nome attributo: tipo dati = val._iniz.

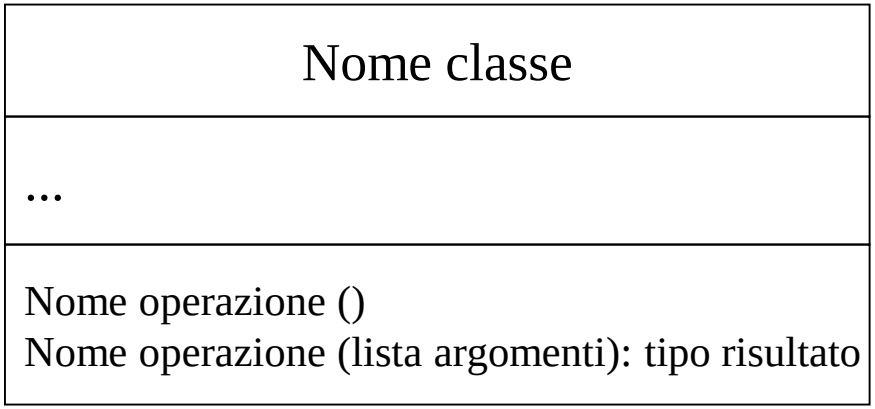
Dialler
- digits : Vector - nDigits : int
+ digit(n : int) # recordDigit(n : int) : boolean

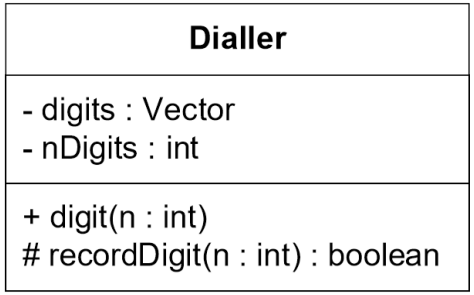
```
public class Dialler
{
  private Vector digits;
  int nDigits;
  public void digit(int n);
  protected boolean recordDigit(int n);
}
```

- Gli attributi sono in genere considerati *private*

Operazioni

- Rappresentano le **azioni** che si possono compiere sugli oggetti di una classe
  - L'insieme delle operazioni determina la **responsabilità** della classe
- In un linguaggio OO, corrispondono a un **metodo** della classe






```
public class Dialler
{
   private Vector digits;
   int nDigits;
   public void digit(int n);
   protected boolean recordDigit(int n);
}
```

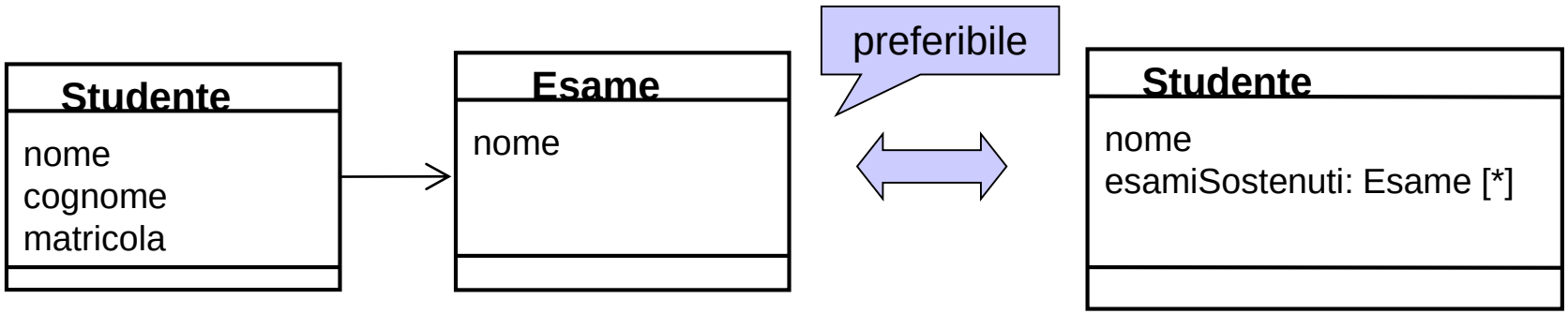
- Le operazioni possono essere *public* (+), *protected* (#) o *private* (-)
- In genere, nel diagramma non si mostrano le operazioni semplici di accesso a un attributo, in lettura (getAtt1) o scrittura (setAtt1)

Responsabilità di una classe

Classe	Responsabilità
Sale	• Sa calcolare il totale della vendita • Conosce le righe di vendita della vendita
SalesLineItem	• Sa calcolare il totale parziale della riga di vendita • Conosce il prodotto della riga di vendita
ProductDescription	• Conosce il prezzo del prodotto

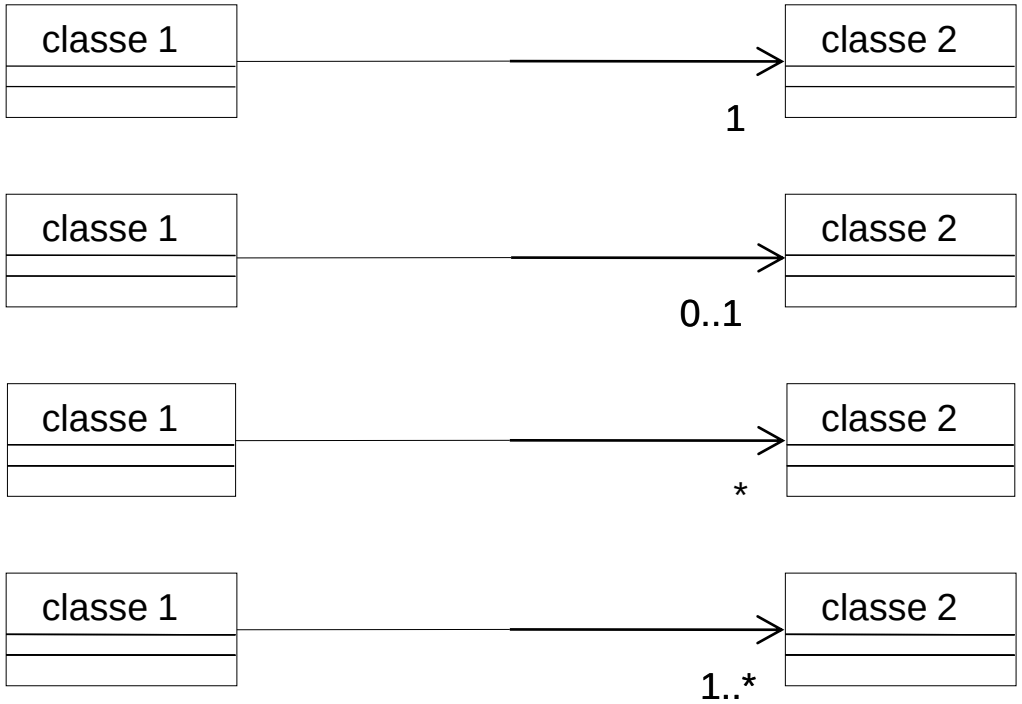
Associazioni

- Modo alternativo per rappresentare **proprietà strutturali** il cui tipo base è un'altra classe del sistema
- Le associazioni hanno un verso di navigazione (freccia aperta) che indica a quale classe si riferisce la proprietà rappresentata come associazione



Molteplicità delle associazioni

- La molteplicità specifica quante istanze di una classe possono essere associate con una singola istanza di un'altra classe
- Il simbolo di molteplicità adiacente alla classe 2 indica quante istanze della classe 2 possono avere legami con una singola istanza della classe 1



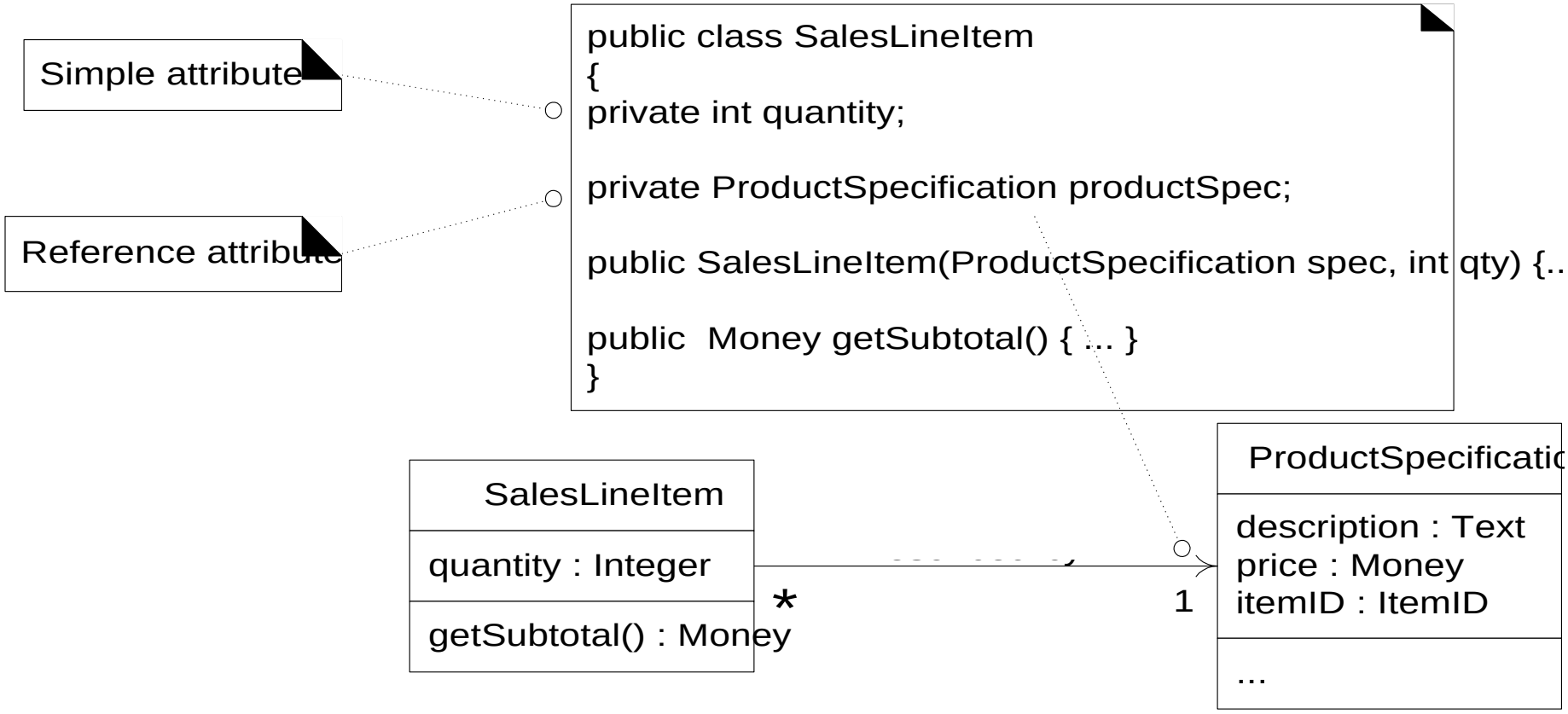
uno ed uno solo

opzionale (zero o uno)

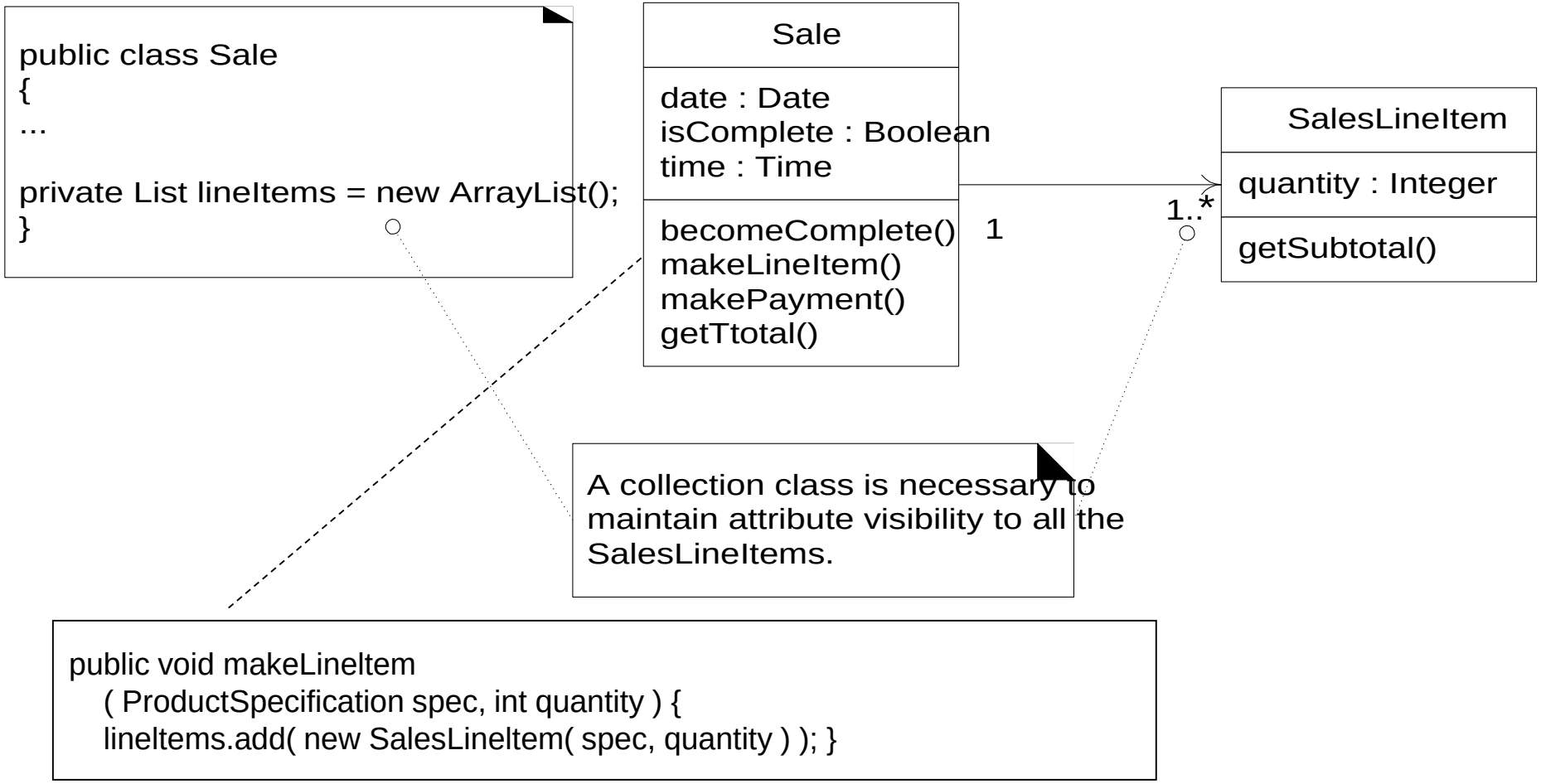
molti (zero o piu' di uno)

uno o piu' di uno

Realizzazione di un'associazione a molteplicità singola

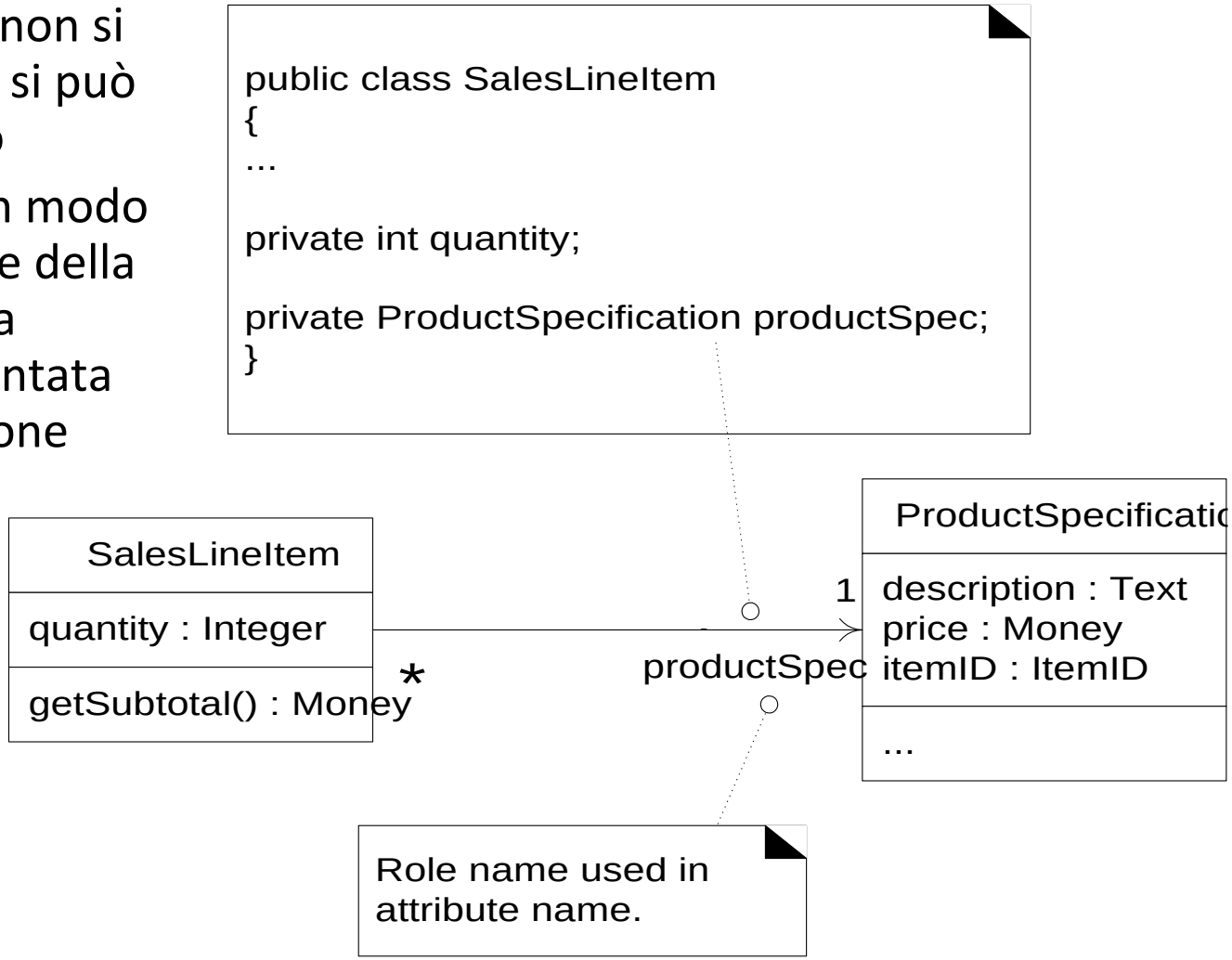


Realizzazione di un'associazione a molteplicità multipla



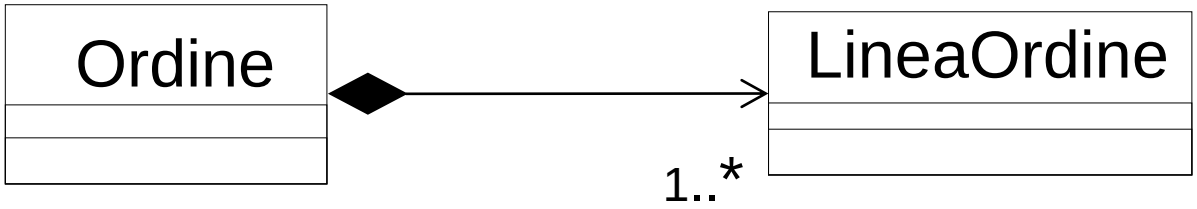
Ruoli

- Le associazioni non si etichettano ma si può indicare il ruolo
- Il ruolo indica in modo esplicito il nome della proprietà di una classe rappresentata come associazione



Aggregazione e composizione

- L'aggregazione non ha un significato distinto da una generica associazione e quindi non c'è bisogno di usarla
- La **composizione è un'associazione** in cui:
  - L'oggetto contenitore è l'unico responsabile della creazione degli oggetti contenuti
  - La cancellazione dell'oggetto contenitore ha come effetto la cancellazione degli oggetti contenuti
  - La copia dell'oggetto contenitore ha come effetto la copia degli oggetti contenuti



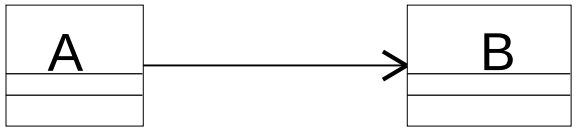
Dipendenze

- Si ha una dipendenza tra classi (in generale tra classificatori) se **una classe ha conoscenza di un'altra classe**
  - Il cambiamento in una classe può causare un cambiamento in un'altra classe che dipende da essa
  - Vale anche per le associazioni e le generalizzazioni
- Una dipendenza rappresenta un rapporto cliente-fornitore mediante il quale le classi collaborano
- L'uso più comune è per indicare una **relazione temporanea tra classi**

Dipendenze e visibilità

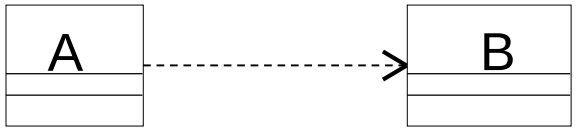
Come nelle associazioni, l'oggetto fornitore deve essere visibile all'oggetto cliente

- Associazione




```
public class A {
  private B itsB;
  public void f() {
    itsB.f();
  }
}
```

- Dipendenza



  - Per creazione
    - L'oggetto fornitore è creato in un'operazione da un'operazione dell'oggetto cliente

```
public class A {
  public B makeB() {
    return new B();
  }
}
```

  - Per passaggio di parametri
    - L'oggetto fornitore è un parametro di un'operazione dell'oggetto cliente

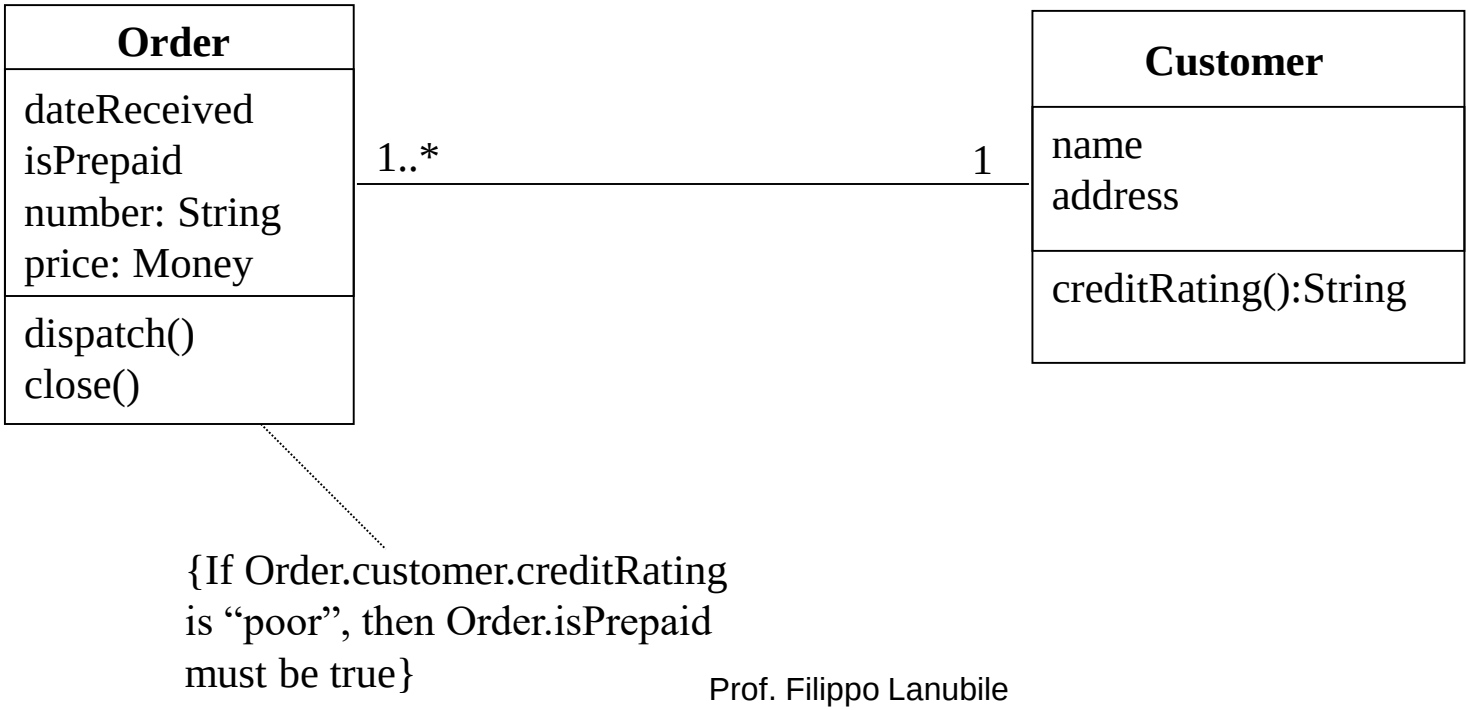
```
public class A {
  public void f(B b) {
    // use b;
  }
}
```

  - Per variabile locale
    - l'oggetto fornitore è dichiarato come variabile locale di un'operazione dell'oggetto cliente
  - Per nome globale o metodo statico
    - l'oggetto fornitore è globale rispetto al cliente

```
public class A {
  public void f() {
    B b = new B();
    // use b
  }
}
```

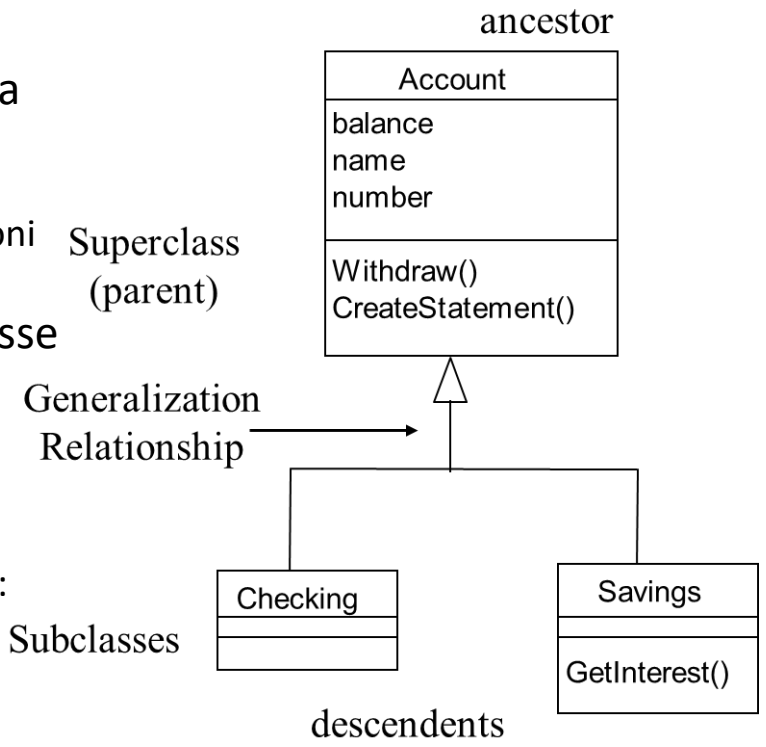
Vincoli

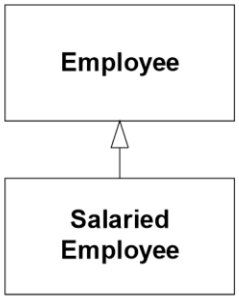
- Rappresentano ulteriori restrizioni applicabili agli elementi di un diagramma delle classi
- Si possono esprimere in modo informale tra parentesi graffe
- Object Constraint Language (**OCL**) è un linguaggio formale per specificare vincoli



Generalizzazione

- La relazione di generalizzazione rappresenta un meccanismo di condivisione delle caratteristiche delle classi
  - Caratteristiche: operazioni, attributi, associazioni
- Gli oggetti delle classi specializzate (**sottoclassi**) condividono (**ereditano**) le stesse caratteristiche della classe generale (**superclasse**)
- Una sottoclasse può:
  - definire caratteristiche aggiuntive: **estensione**
  - modificare l'implementazione delle operazioni: **overriding**



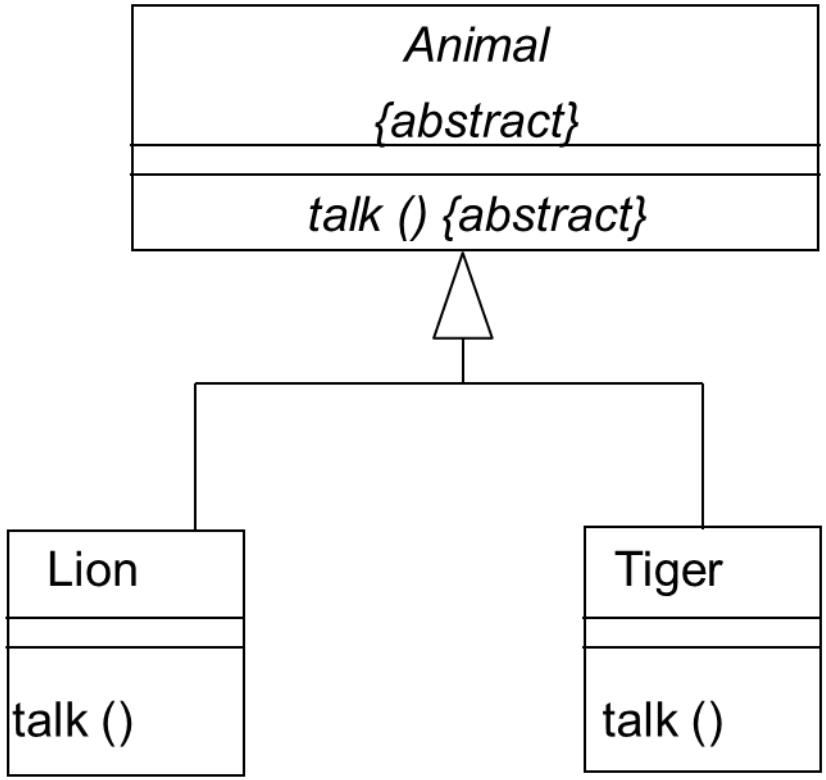



```
public class Employee
{
   ...
}

public class SalariedEmployee extends Employee
{
   ...
}
```

Classi astratte e concrete

- Una classe astratta è una classe che non può essere direttamente istanziata
  - Contiene una o più **operazioni astratte** (non hanno implementazione)
- Per istanziare una classe astratta occorre prima creare una **classe concreta** (può essere direttamente istanziata)
  - E' dichiarata come sottoclasse di una classe astratta
  - La classe concreta aggiunge l'implementazione alle operazioni astratte



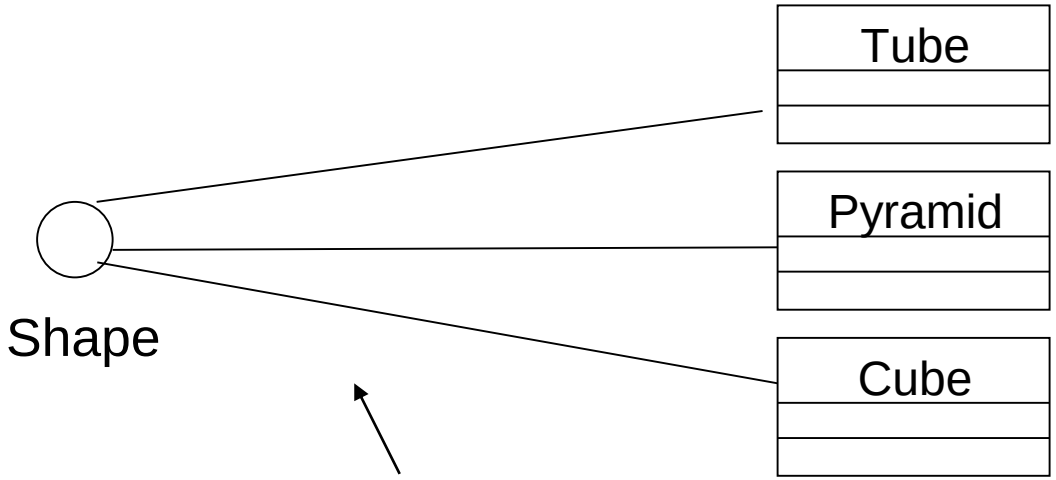
Interfacce

- Un'interfaccia è una classe priva di implementazione
  - Equivale a una **classe astratta dotata solo di operazioni astratte**
  - Molti linguaggi OO hanno un costrutto specifico
- Un'interfaccia può estendere una o più interfacce (relazione di generalizzazione)
  - Supertipo e sottotipo: es. Container e List
- Una classe **fornisce (realizza**) un'interfaccia se è sostituibile a essa (perché ne implementa una o più operazioni astratte)
  - Es. private List lineItems = new ArrayList();
- Una classe **richiede** un'interfaccia se necessita delle sue operazioni per funzionare
  - Ha una dipendenza da essa
  - Es. class Order {

```
…
public void addLineItems(LineItem arg) {
lineItems.add(arg);
}
```

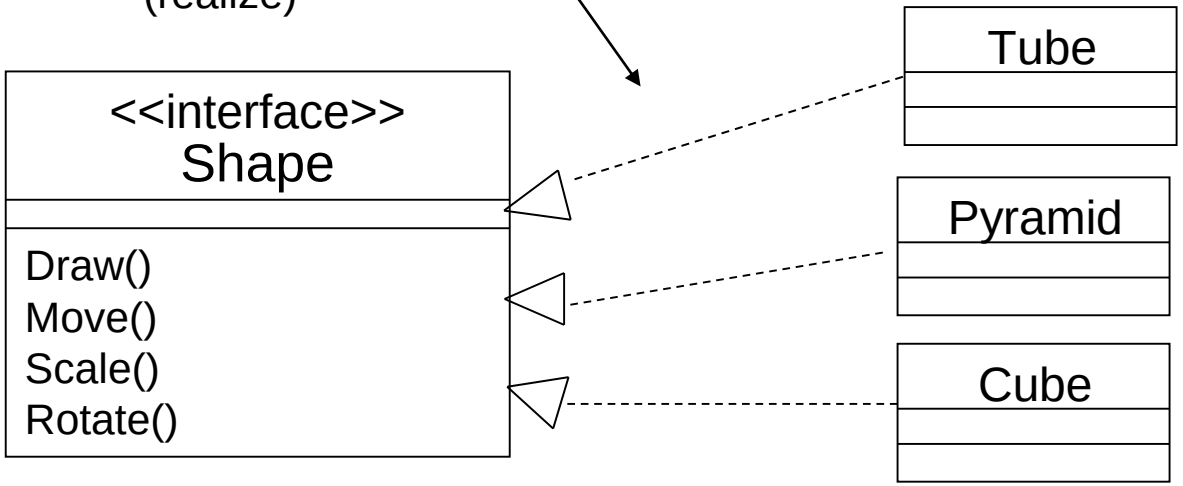
Rappresentazione di un'interfaccia

Rappresentazione iconica ("lollipop")

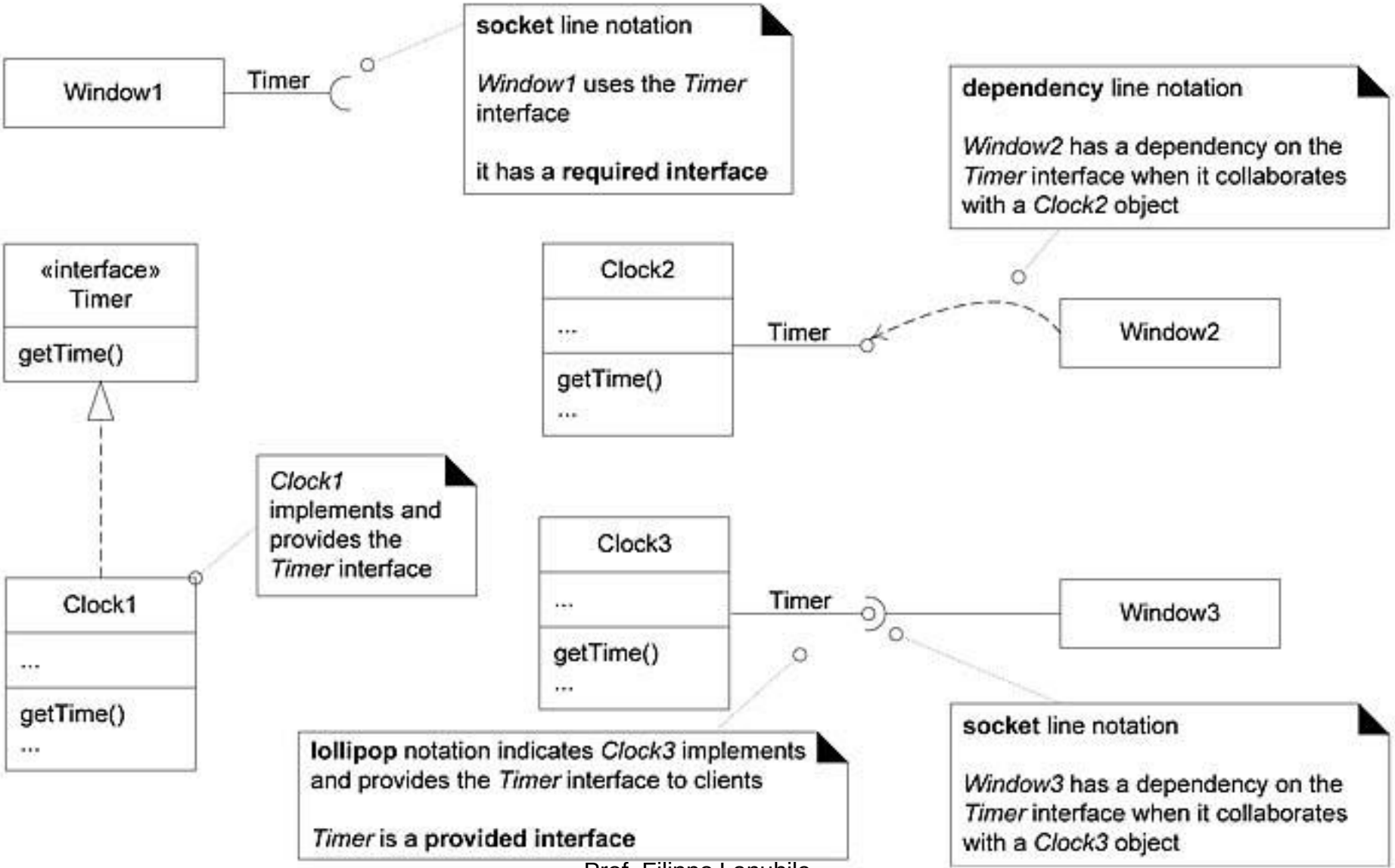


Relazione di realizzazione (realize)

Rappresentazione canonica (Class/Stereotype)



Diverse notazioni per mostrare le interfacce in UML



Costruzione di diagrammi delle classi con prospettiva software

- Non cercare di mettere tutto su un solo diagramma
- Ogni diagramma deve avere uno scopo
  - Diagramma che mostra le classi che collaborano nella realizzazione di una user story
  - Diagramma che mostra una gerarchia di classi
  - Diagramma che mostra le classi di un sottosistema (package)
- Non è necessario mostrare in un diagramma tutte le caratteristiche di una classe
  - solo quelle considerate significative per il diagramma in questione