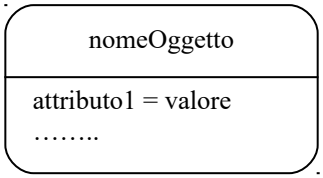
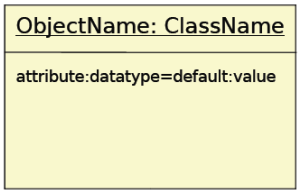
Descrizione di un **oggetto** con **UML** (**diagramma degli oggetti**):

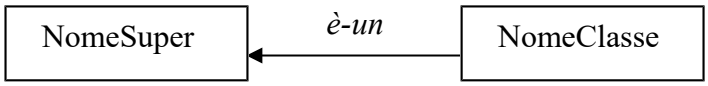
- **nome**
- ***stato*** (valori degli attributi)



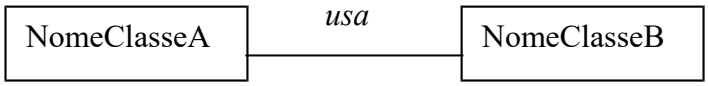


Descrizione delle possibili **relazioni tra classi** con **UML** :

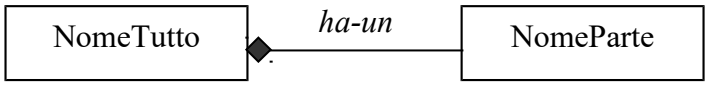
- NomeClasse
- linea di collegamento con possibile ***etichetta*** esplicativa, ***molteplicità*** e navigabilità (freccia disegnata a fianco dell'etichetta per indicare un verso privilegiato); la **forma di una delle estremità** specifica il tipo di relazione (e può essere considerata una forma di navigabilità):
  - specializzazione (**ereditarietà**): una freccia orientata verso la "Superclasse"



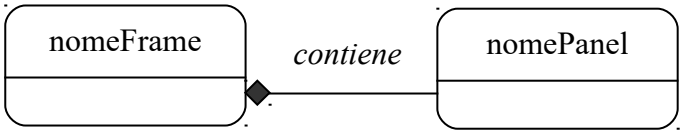
  - **associazione**: nessun simbolo. Si manifesta se c'è un collegamento tra gli oggetti delle due classi che si scambiano messaggi (richiamo di metodi pubblici con ambito di istanza e/o di classe) come unica[1] relazione.



  - relazione di tipo *contenimento*: un piccolo rombo  vuoto se **aggregazione**, pieno  se **composizione**



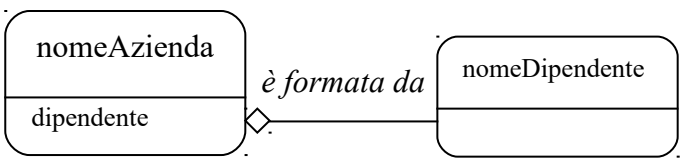
Esempi con gerarchia di oggetti di classi in relazione *whole-part* (tutto-parte):



La parte non esiste senza il tutto.

Implementando la creazione dell'oggetto di altra classe all'interno del costruttore della classe principale si stabilisce una **dipendenza necessaria**. Un pannello non esiste senza un frame che lo contiene.

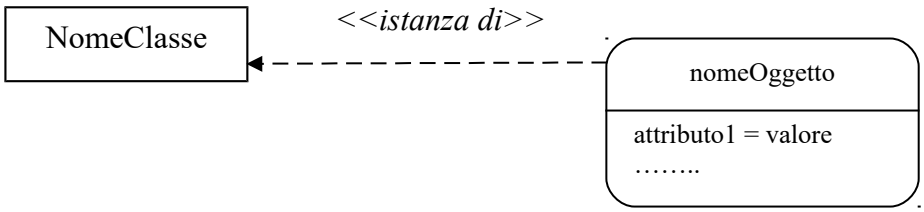
Quando, invece, l'attributo di una classe è una variabile di riferimento per l'altra, si stabilisce una dipendenza di tipo aggregazione :



Gli oggetti della classe principale **possono** creare zero, uno o più oggetti dell'altra. Le parti possono esistere anche senza il tutto.

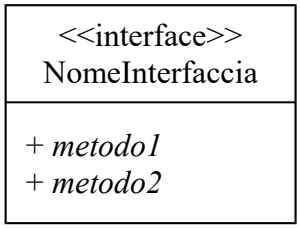
[1] Ogni tipo di relazione coinvolge lo scambio di messaggi tra oggetto fornitore di servizi ed oggetto fruitore, quindi l'associazione potrebbe essere considerata la **relazione generica** da cui derivano quelle di contenimento come forme particolari. Il concetto più generale di dipendenza esprime anche solo il "venire in contatto" cioè "influenzare".

In un **diagramma degli oggetti**, l'**UML** considera un oggetto collegato alla sua classe come una forma di relazione (relazione di **realizzazione** individuata da linea tratteggiata, con una freccia dalla parte della classe e lo stereotipo <<*istanza di*>>) :

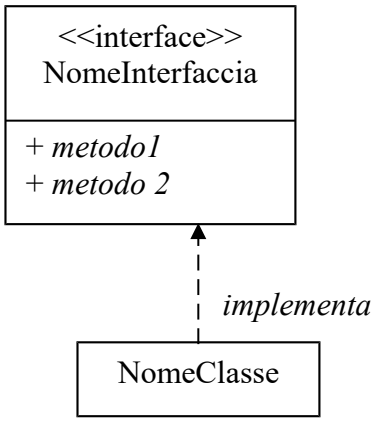


In un **diagramma delle classi**, l'**UML** prevede di rappresentare un'**interfaccia** (insieme di operazioni che una classe offre ad altre classi, utile per raggruppare tale collezione di metodi comuni a più classi, quando non sono tutte correlate ad una particolare classe padre) con i seguenti simboli:

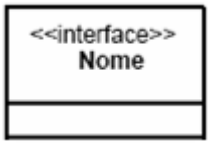
- uso dello stereotipo <<interface>>
- nessun attributo (Java modifica tale indicazione e permette di dichiarare attributi costanti con ambito di classe cioè ***static final***)
- tutti i metodi **pubblici** (nome preceduto da +)
- tutti i metodi **astratti** (scritti in corsivo): dichiarati solo con intestazione e non implementati



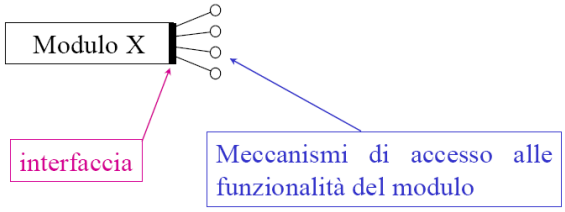
L'**UML** permette di rappresentare una classe che **implementa** un'**interfaccia** (cioè si comporta come dichiarato dall'interfaccia) con una relazione di **realizzazione** (linea tratteggiata con una freccia dalla parte dell'interfaccia ed eventuale etichetta):



In alternativa, l'implementazione dell'interfaccia può essere indicata con un **piccolo cerchio vuoto**



in analogia alla modellazione dell'**interfaccia di una classe** (caratteristiche visibili esternamente: fatto i metodi pubblici). L'*information hiding* e l'*incapsulamento* separano l'interfaccia di una classe dalla sua rappresentazione interna (implementazione/realizzazione)

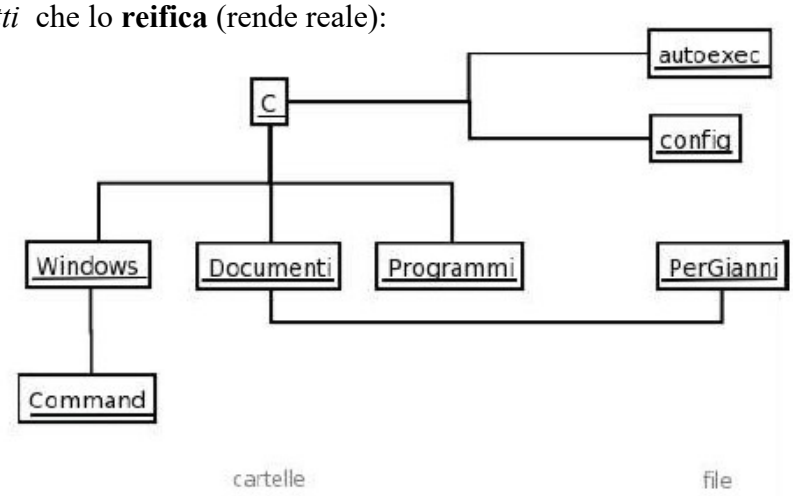


di

Associazione riflessiva: se una classe ha un'associazione con sé stessa

Esempio con diagramma di classe

e relativo diagramma degli oggetti che lo **reifica** (rende reale):

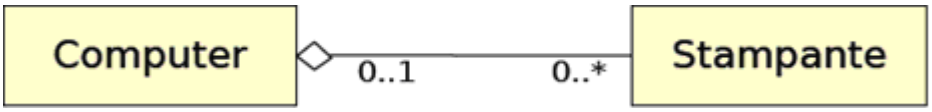


Esempi di **molteplicità**:

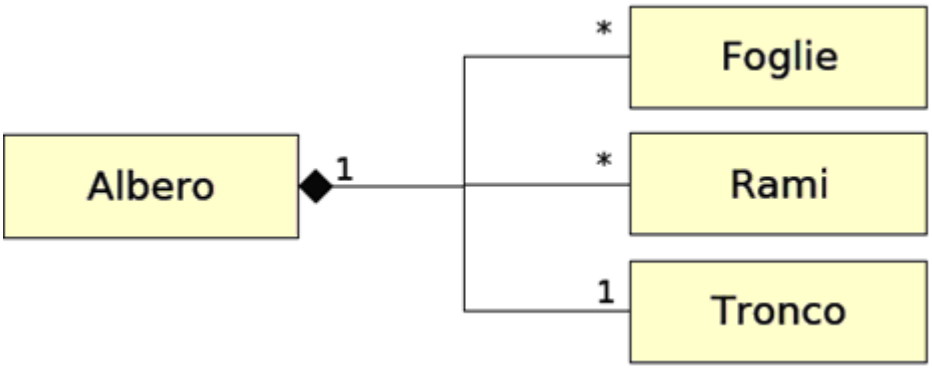
- associazione **uno-a-uno** (bidirezionale):

- associazione **molti a uno** (bidirezionale):

- aggregazione



- composizione



Appendice

Simboli:

• Generalizzazione: relazione tassonomica tra un elemento più generale e un elemento più specifico (**ereditarietà**)

• **Associazione**: scambio di messaggi tra oggetto fornitore di servizi ed oggetto fruitore con navigabilità bidirezionale tra classi

• Commento, associazione tra classe e associazione, integrazione della sintassi

• Dipendenza: nella relazione tra due elementi di modellazione, in cui una modifica a un elemento (detto "indipendente") ha effetti sulla modellazione dell'altro elemento (detto "dipendente")
Es: dipendenza semantica per illustrare "istance of" oppure "uso"

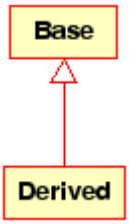
• **Realizzazione**: relazione tra una specifica e la sua implementazione

• **Associazione qualificata** nella quale un oggetto di un classe è identificato dalla classe associata mediante una"***chiave primaria***".

Molteplicità : uno (1)
numero definito (N)
"zero o più" (*) anche (0..*)
da 1 a N (1..N)
"uno o più" (1..*)

Tipi di relazioni tra classi:

Ereditarietà



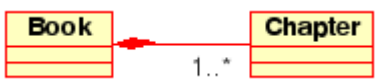
Associazione

una compagnia può assumere "uno o più" dipendenti

Aggregazione (implementata con uso di array dinamico in Java)

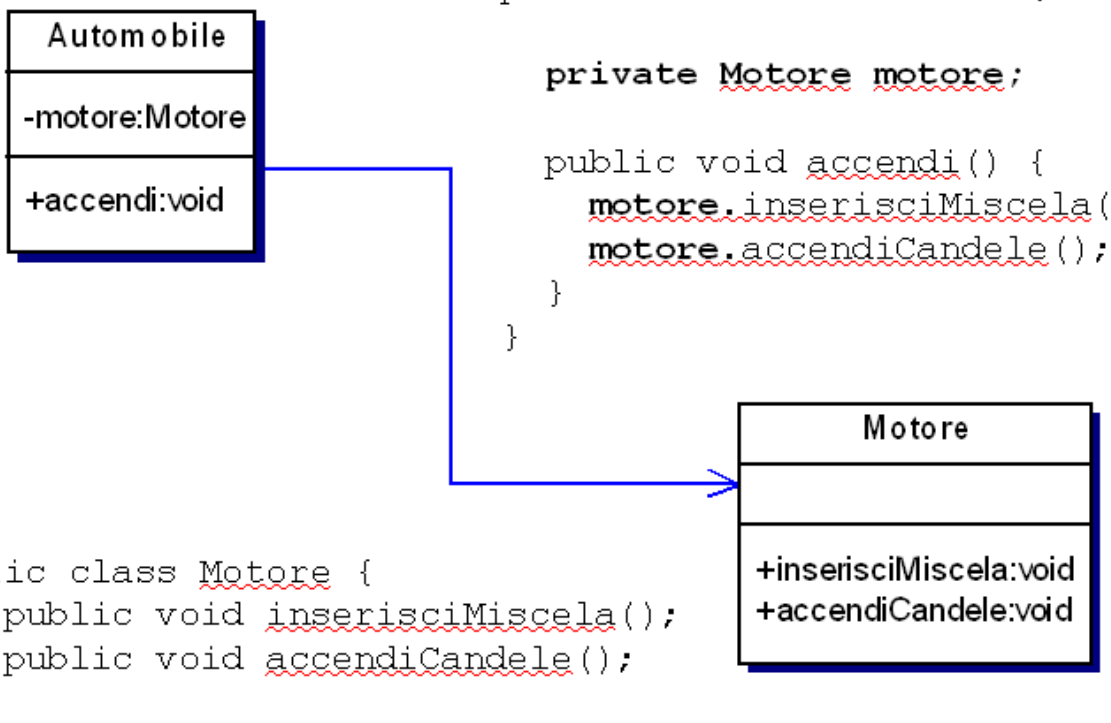
la ruota può esistere senza auto

Composizione



uno o più capitoli non possono esistere senza libro
la distruzione del tutto determina la distruzione delle parti

In alcuni casi una relazione di **associazione** (unidirezionale) evidenzia la "*navigabilità*" con una direzione ad una estremità del collegamento. Un'associazione, in Java, viene solitamente implementata con un ***reference***

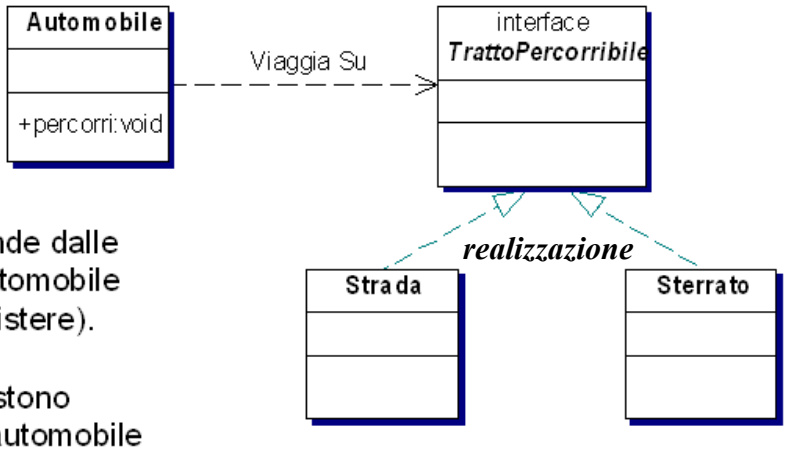



```
public class Automobile {

   private Motore motore;

   public void accendi() {
     motore.inserisciMiscela();
     motore.accendiCandele();
   }
}
```

```
public class Motore {
    public void inserisciMiscela();
    public void accendiCandele();
}
```

La generica **dipendenza** indica che un determinato oggetto può, in certe circostanze, chiamare i metodi di un altro pur senza possederne un'istanza. La classe dipendente presuppone l'esistenza della classe da cui dipende; non vale il viceversa. L'oggetto dipendente, in Java riceve un'istanza dell'oggetto da cui dipende come argomento di una chiamata a metodo:

```
public class Automobile {

  public void percorri(TrattoPercorribile p) {
  ....
  }
}
```



Nota: L'automobile dipende dalle strade (senza strade l'automobile non avrebbe senso di esistere).

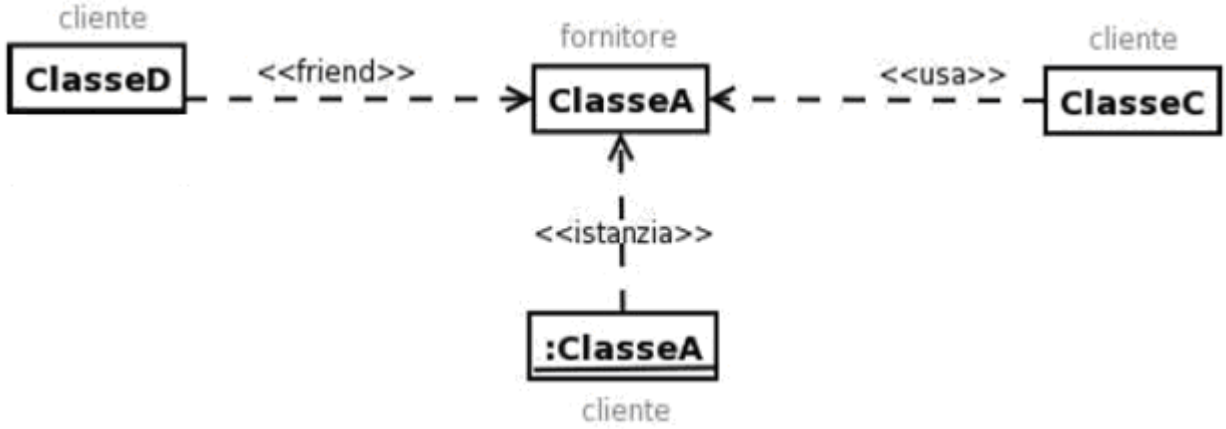
Al contrario le strade esistono indipendentemente dall'automobile

Relazioni di **dipendenza** dove un cambiamento a un elemento (detto "fornitore") può influenzare l'altro elemento (il "cliente") possono essere di tipo:

• **Uso**: *il cliente utilizza alcuni dei servizi resi disponibili dal fornitore per implementare il proprio comportamento*
• **Astrazione**: *una relazione tra cliente e fornitore in cui il fornitore è più astratto del cliente*
• **Accesso**: *il fornitore assegna al cliente un qualche tipo di permesso di accesso al proprio contenuto*
• **Binding**: *il fornitore assegna al cliente dei parametri che saranno istanziati a valori specifici dal cliente (processo che* lega/associa *la definizione cioè il codice del metodo alla chiamata: se* ***statico*** *avviene in compilazione; se* ***dinamico*** *al run-time)*

I relativi stereotipi possono essere:

• Uso: <<usa>>, <<chiama>>, <<parametro>>, <<invia>>, <<istanzia>>
• Astrazione: <<origine>>, <<rifinisce>>, <<deriva - da>>
• Accesso: <<accede>>, <<importa>>, <<friend>>
• Binding: <<bind>>

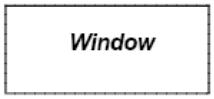


nb: il **modello statico** di riferimento per la modellazione strutturale con **UML** permette di mostrare:

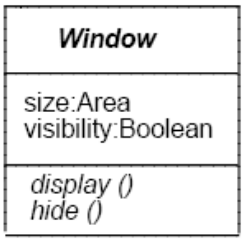
- le entità presenti nel modello (classi, interfacce, componenti, nodi)
- la struttura interna
- le relazioni statiche tra entità

il livello di astrazione del diagramma (**struttura statica**) può essere:

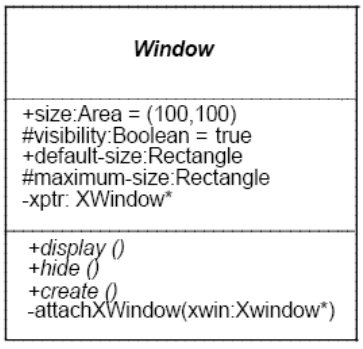
- concettuale



- di specifica



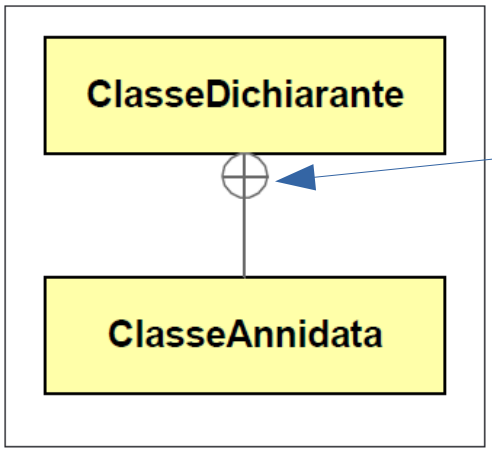
- di implementazione



Classe annidata

Classi annidate (***nested class***) o interne (***inner class***) cioè definite all'interno della definizione di un'altra classe. Accessibili solo alla classe esterna in cui è annidata e agli oggetti di quella classe

Rappresentazione di classi annidate.



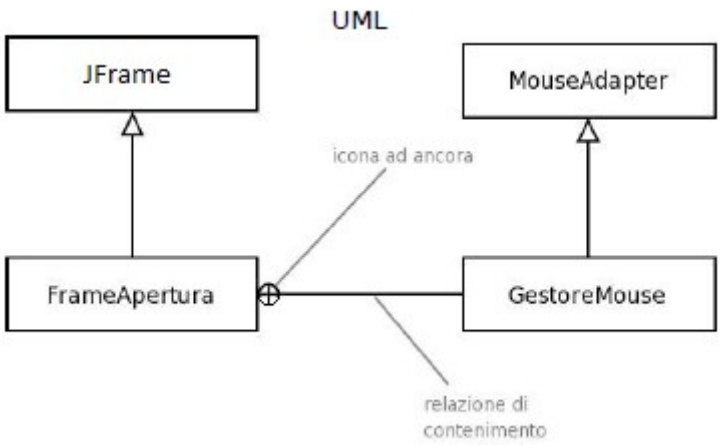
Se si vuole rappresentare tale relazione in diagrammi UML in modo meno generico si può utilizzare una icona ad ancora come illustrato in figura

icona ad ancora (cerchio crociato)

indica che la classe che si trova di fianco al cerchio crociato **contiene** la classe che si trova all'altra estremità della linea

Esempio: si definisce una classe **interna** all'applicazione di tipo Gui

```
class FrameApertura extends JFrame {

  // creazione oggetti grafici

  private class GestoreMouse extends MouseAdapter{

      // possibilità d'uso degli oggetti grafici
     // definiti nella classe esterna

  } // fine nested class

}// fine class esterna
```



Riferimenti online:

http://www.cs.unibo.it/cianca/wwwpages/labspo/Lab3.pdf pg.44

Cambiamenti da UML 1.3 a UML 1.4 (pg.26-27) ed evoluzione (pg.28) o altro (pg.12)

Ultima versione 2.5 ed altre pagina dedicata su wikipedia