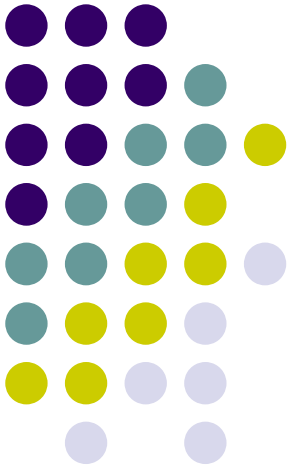
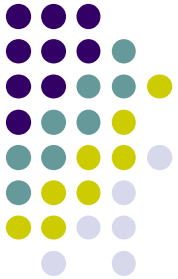
Corso di Programmazione

Contenimento e Associazione

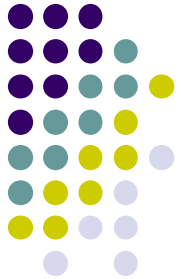


Contenimento



- La composizione è una relazione tra classi
- Realizza il concetto “tutto/parte di”
- Può essere indicata con lo “slogan”: *has-a*
- La relazione di composizione può essere realizzata fondamentalmente in due modi:
  - Composizione (o contenimento stretto)
  - Aggregazione (o contenimento lasco)

Contenimento stretto

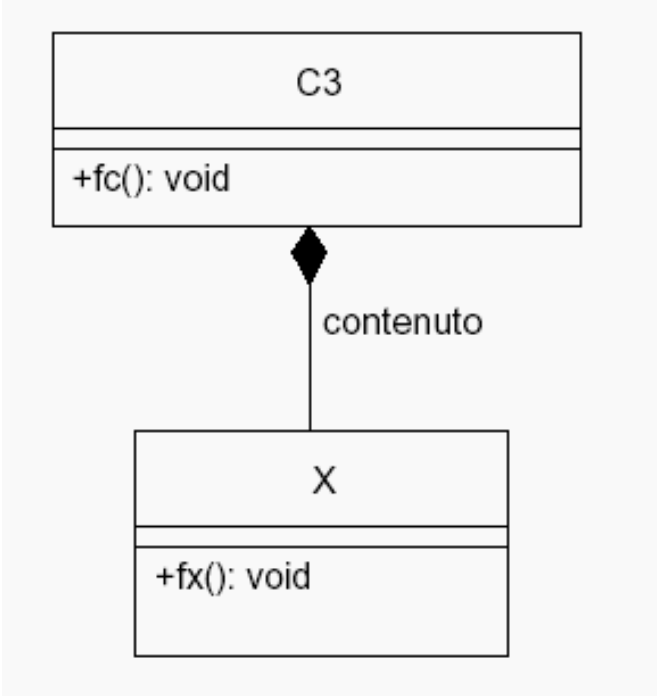


- La relazione di contenimento stretto indica, concettualmente, che l'oggetto contenuto non ha una vita propria ma esiste in quanto parte dell'oggetto contenitore.
- L'oggetto contenuto esiste solo per realizzare una parte dell'oggetto contenitore

Contenimento stretto: realizzazione

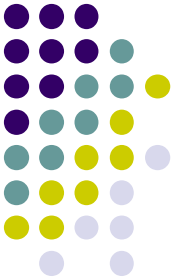
- Il ciclo di vita dell'oggetto contenuto è legato al ciclo di vita del contenitore
  - Il contenimento stretto si realizza introducendo nella classe **contenitore** C un attributo x della classe **contenuto X**.
  - **L'utente** della classe C ha la responsabilità di istanziare un oggetto di tipo **contenitore**, richiamando il suo costruttore con tutti i valori di inizializzazione necessari sia all'inizializzazione del **contenitore** che del **contenuto**.

Contenimento stretto: diagramma delle classi



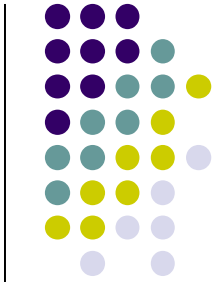
- C3 è la classe contenitore
- X è la classe di cui un oggetto è contenuto come membro nella classe C3

Contenimento lasco



- La relazione di contenimento lasco indica *l'indipendenza* del ciclo di vita dell'oggetto **contenuto** dall'oggetto **contenitore**
- L'oggetto interno esiste indipendentemente dal suo ruolo di "parte di"
- Questo comporta la non responsabilità da parte del **contenitore** per la creazione e distruzione dell'oggetto **contenuto**.

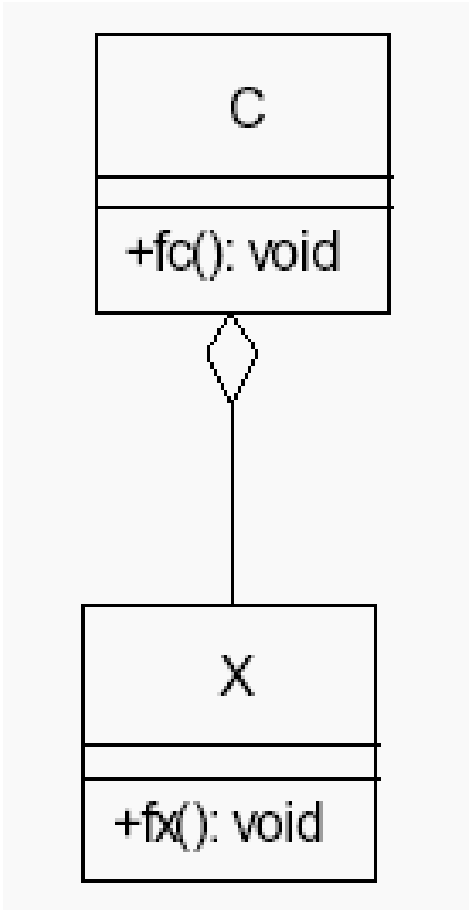
Contenimento lasco: realizzazione



- **Il contenimento lasco** si realizza introducendo nella classe **contenitore C** un riferimento/puntatore all'oggetto **x contenuto**. ***Il costruttore*** di C riceve in ingresso il puntatore all'oggetto contenuto. **L'utente** della classe C ha la responsabilità di:
  - creare l'oggetto contenuto.
  - definire ed inizializzare un riferimento/puntatore ad esso.
  - costruire l'oggetto contenitore passando il puntatore al contenuto.

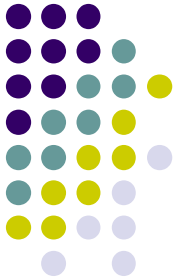
- Il contenitore dovrà definire un costruttore che riceva in input il **puntatore** (o il **riferimento**) all'oggetto contenuto.

Contenimento lasco: diagramma delle classi



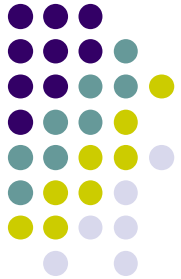
- C è la classe contenitore
- X è la classe “contenuta”

Associazione tra Classi



- L'associazione tra due (o più) classi è una relazione che esprime un legame semantico tra le classi.
- Essa è caratterizzata da:
  - ✓ un **nome**: esprime il legame semantico tra le classi associate
  - ✓ un eventuale **ruolo** la cardinalità delle connessioni giocato dalle parti associate
  - ✓ la **molteplicità** dell'associazione: esprime tra oggetti "istanze" delle classi associate e puo' essere di tipo
    - uno a uno
    - uno a molti
    - molti a molti
  - ✓ la **direzionalità** (uni o bidirezionale): specifica il verso di percorrenza della relazione ed attribuisce inoltre alla classe origine del percorso la responsabilità di tenere traccia dell'associazione

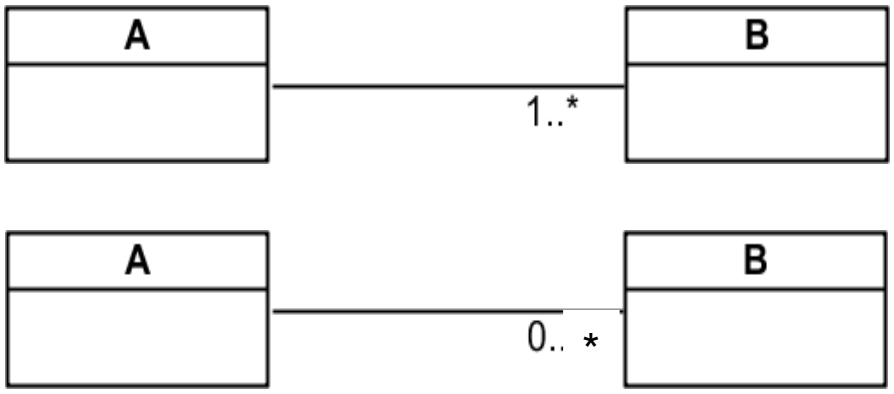
Associazione: realizzazione



- L'associazione si realizza inserendo in A un attributo (riferimento o puntatore) i cui valori sono istanze o collezioni di istanze della classe B e/o viceversa
- Tale attributo è inizializzato da un metodo che quindi ha il compito di «dare inizio» alla relazione
- Analogamente la fine della relazione è in carico ad un altro metodo che «interrompe» il collegamento

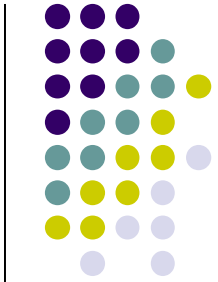
Associazione: diagramma delle classi

- Associazione uno a uno: Il programma ***utente*** deve:
  - Creare i due oggetti
  - Collegarli passando ad entrambi l'indirizzo del partner mediante ***metodi*** previsti allo scopo dalle due classi



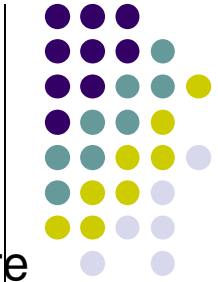
- Associazione uno a molti:
  - Se ad ogni oggetto a di classe A sono associati più oggetti b di classe B l'associazione tra A e B è di uno a molti
  - La classe A partecipa all'associazione mediante una lista *di* riferimenti/puntatori alla classe B
  - La classe A deve fornire i metodi per la rimozione e l'aggiunta delle istanze di B

Implementazione in JAVA

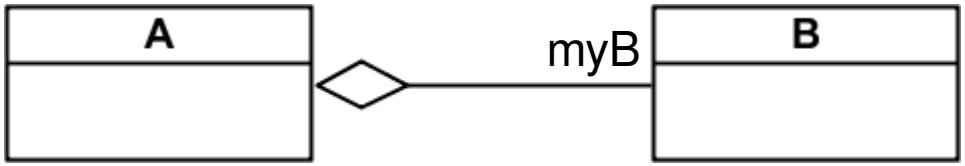


- I concetti finora esposti sono indipendenti dal linguaggio di programmazione
- Una discussione dettagliata della rappresentazione UML delle relazioni di composizione e associazione non è obiettivo di questo corso, ma alcune notazioni verranno introdotte nelle prossime slides quando necessario
- Vediamo ora come si «traducono» le relazioni di composizione e associazione in JAVA

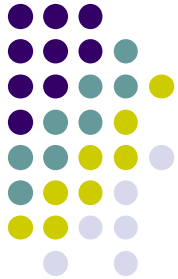
Aggregazione



- In Java una classe può avere attributi che sono *riferimenti* a oggetti di altre classi
- Viene pertanto naturale implementare la contenimento lasco, la cui principale differenza con l'associazione sta nel fatto che tale variabile deve essere inizializzata il prima possibile *quando viene creato l'oggetto contenitore*:
  - L'utente istanzia l'oggetto contenuto e **passa il riferimento al costruttore del contenitore**, che lo utilizza per inizializzare l'attributo che lo riferisce
- Nel diagramma UML il nome della relazione è tradotto con il nome dell'attributo presente nel contenitore



Aggregazione: esempio

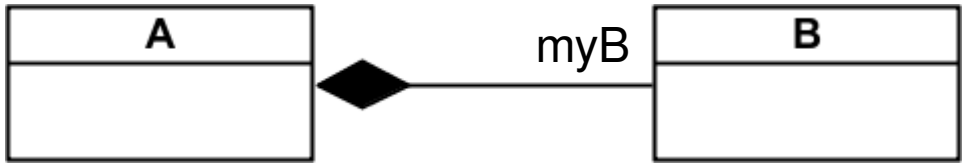


```
public class A {
/*si supponga che B
Sia una classi disponibile */
private B myB;
//costruttore con argomenti di A
public A(B oB){
  myB = oB;
}
```

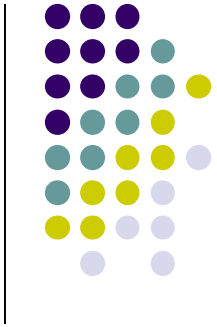
```
public static void
main(String[] args) {
/*alloco al difuori della
  classe A l'oggetto che
  serve per costruire
  l'oggetto a */
B b = new B();
A a = new A(b);
```

Composizione

- Nel caso del contenimento stretto il legame deve essere molto più forte, non si può delegare all'esterno del contenitore la creazione degli oggetti «contenuti»



Composizione: esempio



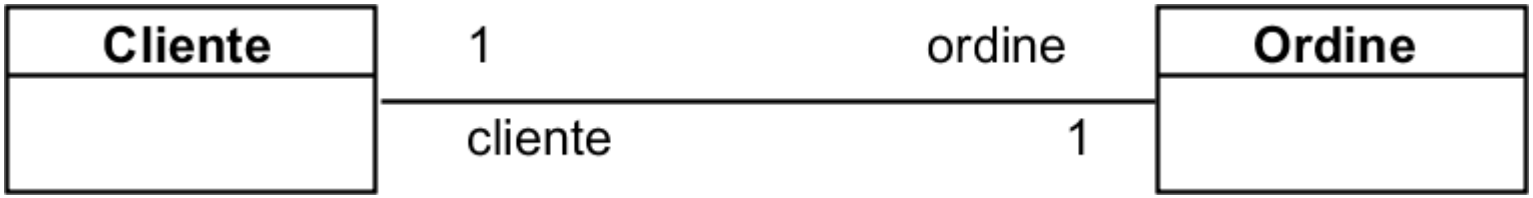
```
public class A {
/*si supponga che B
sia una classe disponibile
i cui attributi sono un intero e
un double */
private B myB;

//costruttore con argomenti di A
public A(int ib, double db){
   myB = new oggettoB(ib,db);

}
```

```
public static void
main(String[] args) {
int nB;
double dB;
A a = new A(nB,dB);
```

Se ci riflettete un attimo
vi accorgerete che questo
lo abbiamo fatto già
tantissime volte...
inconsapevolmente

Associazione tra classi

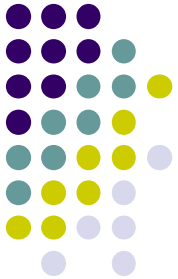
- Anche in queto caso il nome della relazione diventa il nome della variabile relativa all'oggetto

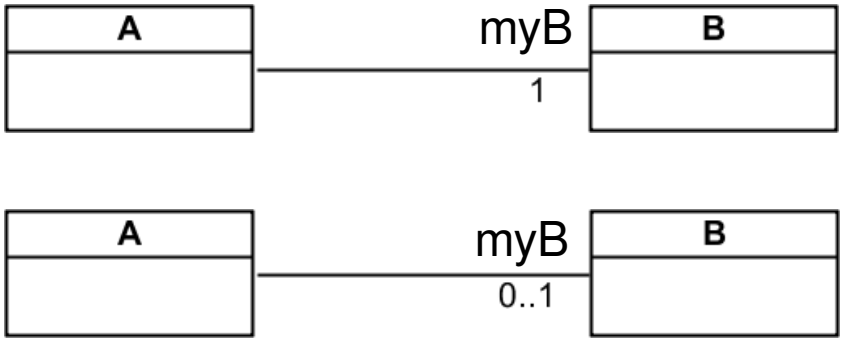



```
public    class Cliente
          { Ordine
          ordine;
…
}
```

```
public    class Ordine
          { Cliente
          cliente;
…
}
```

Associazione tra classi: molteplicità uno a uno



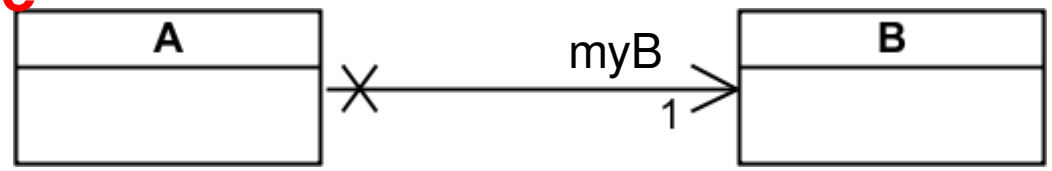



```
publicclass    A
        { BmyB;
…
}
```

- Si traducono allo stesso modo
- Nel secondo caso **la variabile può essere NULL**

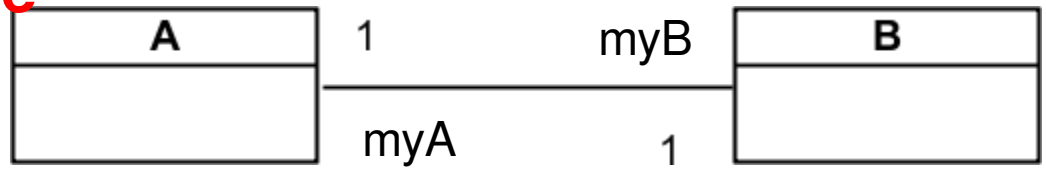
Associazione tra classi uno a uno: navigabilità

Monodirezionale



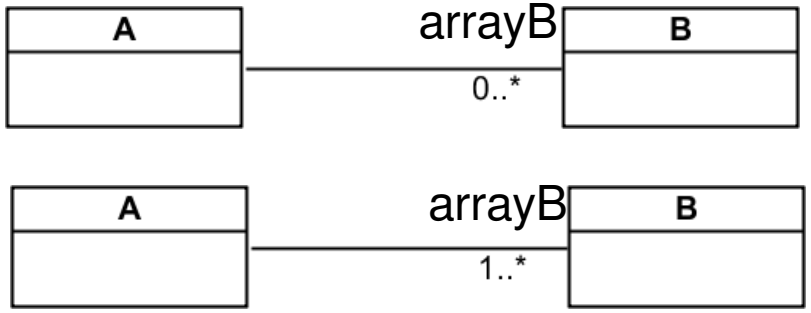

```
publicclass     A
        { BmyB;
…
}
```

Bidirezionale




```
publicclass     A
        { BmyB;
…
}
publicclass     B
        { AmyA;
…
}
```

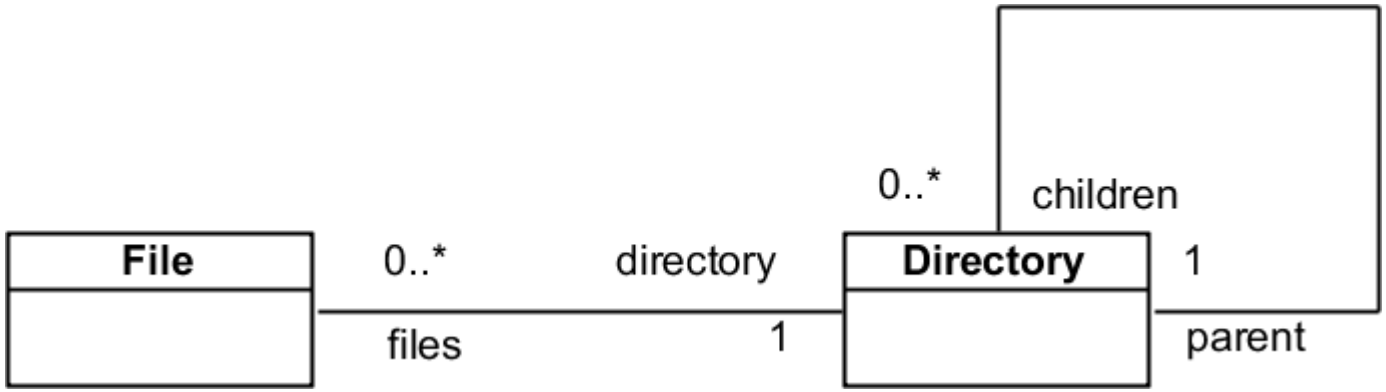
Associazione tra classi: molteplicità uno a molti




```
Public  class  A {
 B[]arrayB;
…
}
```

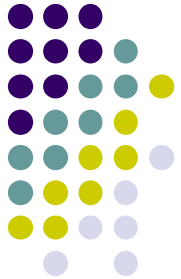
- Si traducono allo stesso modo
- Nel primo caso **l'array può essere vuoto**
- Nel secondo caso **dobbiamo assicurarci che arrayB contenga almeno un riferimento ad un oggetto di B**
- Si possono utilizzare anche altre rappresentazioni per la lista di riferimenti a B

Associazione tra classi: associazione riflessiva

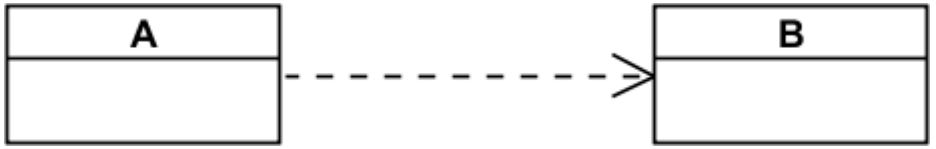



```
public   class      Directory
  {
  File[] files;
  Directory[] children;
  Directory   parent;
…
}
```

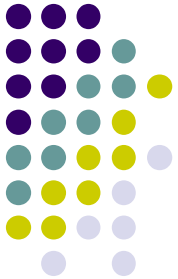
Dipendenze tra classi



- Composizione e associazione sono casi particolari di dipendenza tra classi
- Tra due classi A e B **c'è una dipendenza se**:
  - Un oggetto della classe A invia un messaggio ad un oggetto della classe B, oppure
  - Un oggetto della classe A crea un'istanza della classe B, oppure
  - Un oggetto della classe A riceve un messaggio che ha come argomento un oggetto della classe B
- Una generica dipendenza in UML è raprresentata come in figura:



Dipendenza tra classi 2/3

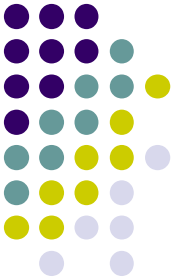


```
public class A {
…
public void  method_A()  {

     B b = new B();
     …
     …
     b.method_B(123);

     }
     …
     }
```

Crea un'istanza della classe B

Manda un messaggio alla classe B

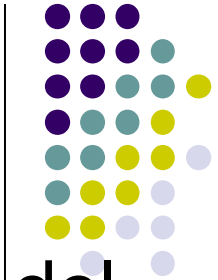
Dipendenza tra classi 1/3



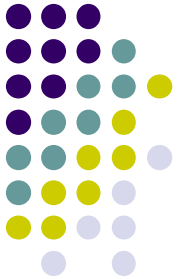
```
public class A {
…
public void  method_A(B  b)
        {
…    c = b.getField();
     …
   }

}
```

Riceve come parametro un oggetto della classe B

Ereditarietà o Contenimento?



- Entrambe consentono un forma di riutilizzo del codice
  - Se i metodi public della classe esistente devono fare parte della interfaccia pubblica della nuova classe deve essere usata l'ereditarietà
  - Il grado di accoppiamento nel contenimento, in particolare nell'aggregazione, è basso. Questo ha dei vantaggi soprattutto perché il forte accoppiamento rende il software più difficile da modificare, questo incentiva l'uso del contenimento



Riferimenti

- Programmare in Java:
  Capitolo 8, §8.8
  Capitolo 9, §9.7