Programmazione Avanzata

relazioni 0,*
composizione – aggregazione
uso list / vector

Pierluigi Roberti

pierluigi.roberti@unitn.it

List vs vector

vector:

- Contiguous memory.
- Pre-allocates space for future elements, so extra space required beyond what's necessary for the elements themselves.
- Each element only requires the space for the element type itself (no extra pointers).
- Can re-allocate memory for the entire vector any time that you add an element.
- Insertions at the end are constant, amortized time, but insertions elsewhere are a costly O(n).
- Erasures at the end of the vector are constant time, but for the rest it's O(n).
- You can randomly access its elements.
- Iterators are invalidated if you add or remove elements to or from the vector.
- You can easily get at the underlying array if you need an array of the elements.

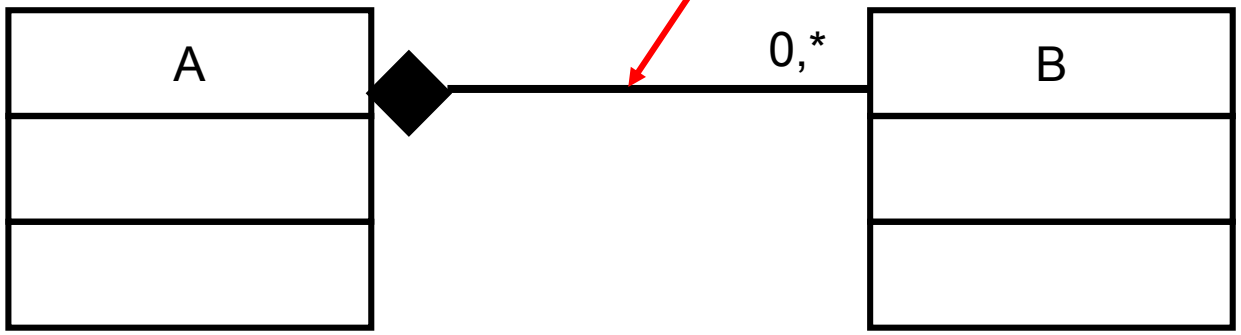
List vs vector

list:

- Non-contiguous memory.
- No pre-allocated memory. The memory overhead for the list itself is constant.
- Each element requires extra space for the node which holds the element, including pointers to the next and previous elements in the list.
- Never has to re-allocate memory for the whole list just because you add an element.
- Insertions and erasures are cheap no matter where in the list they occur.
- It's cheap to combine lists with splicing.
- You cannot randomly access elements, so getting at a particular element in the list can be expensive.
- Iterators remain valid even when you add or remove elements from the list.
- If you need an array of the elements, you'll have to create a new one and add them all to it, since there is no underlying array.

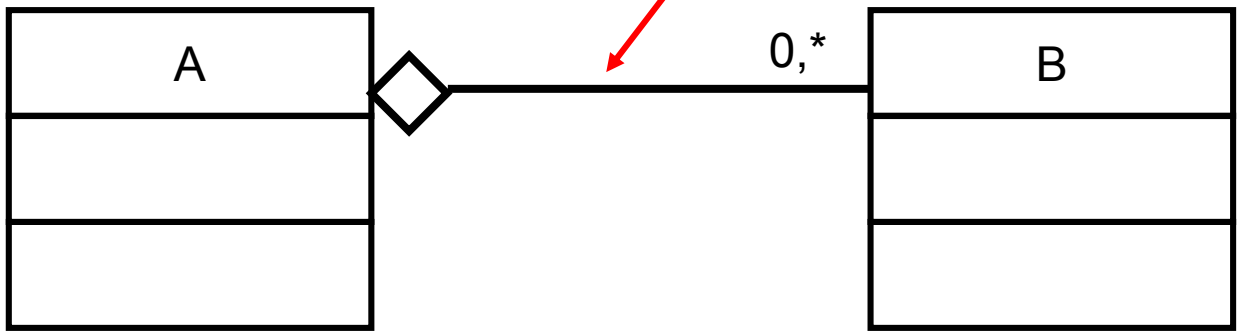
list <B> mialistaB;

composizione



aggregazione

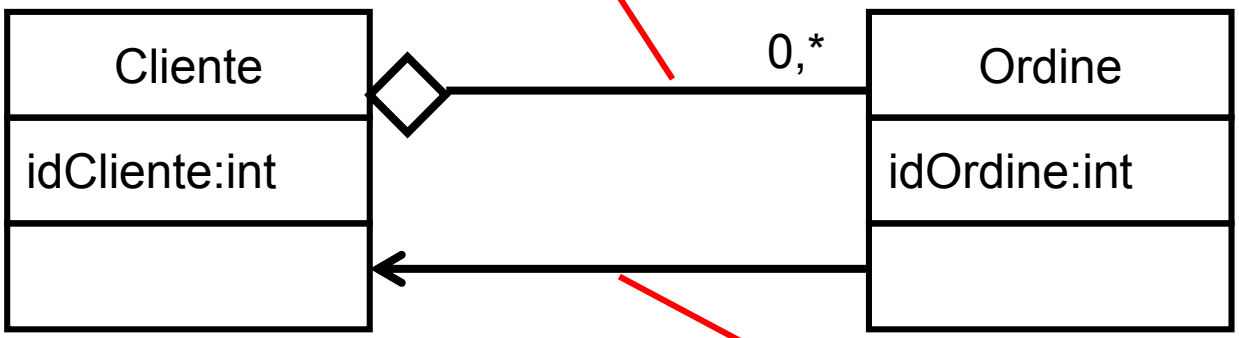
list <B*> mialistaB;



Opzioni

vector <Ordine*> lpo;

list <Ordine*> lpo;



Cliente* pc;