# Design Patterns

Luca Lista

# Design Patterns

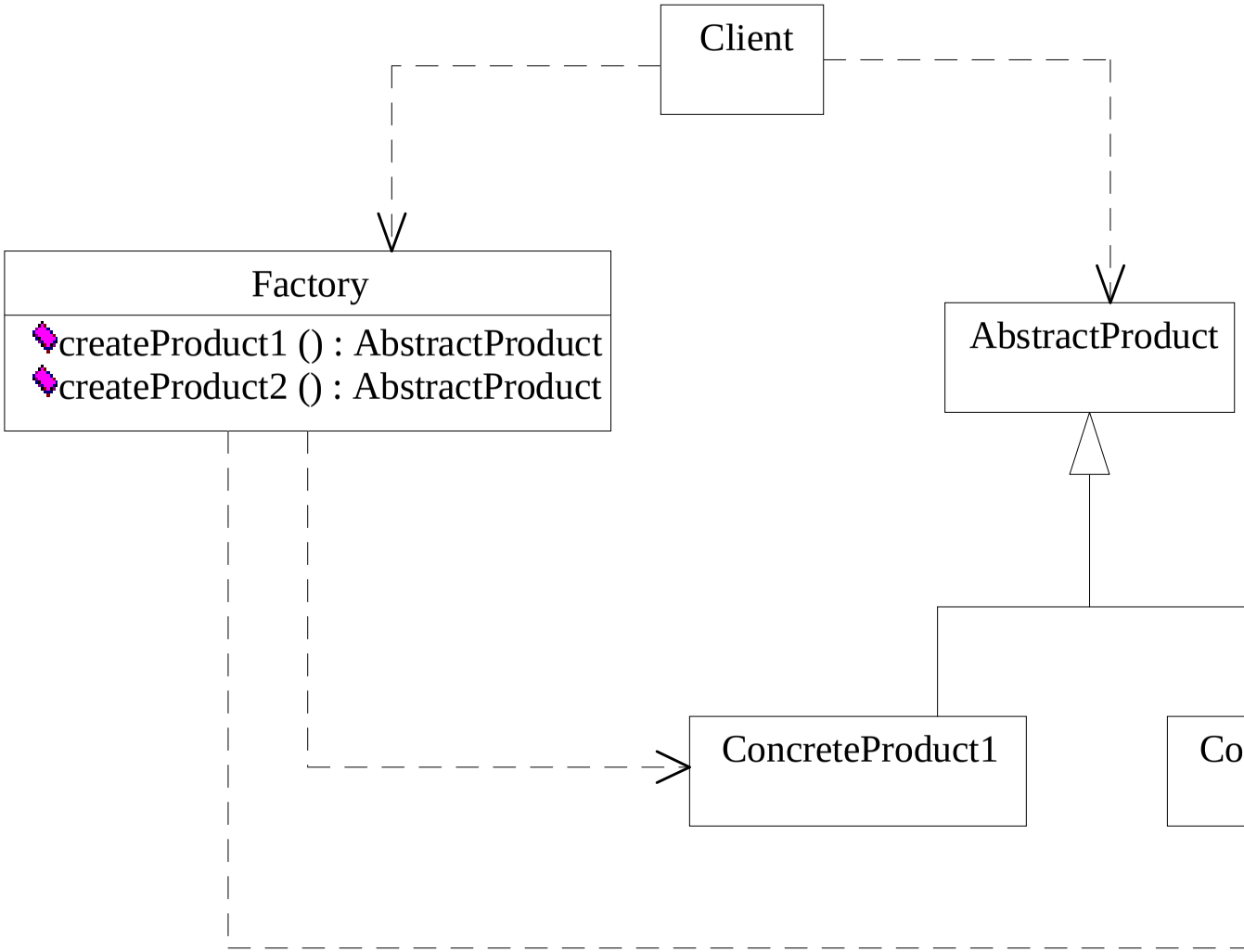
*E. Gamma et al.*, Design Patterns

"Elementi di software OO riutilizzabile"

- Piccoli insiemi di classi che collaborano implementando dei comportamenti tipici
  - Creational patterns
  - Structural patterns
  - Behavioral patterns

Alcuni pattern classici stanno diventanto obsoleti grazie al supporto dei Template

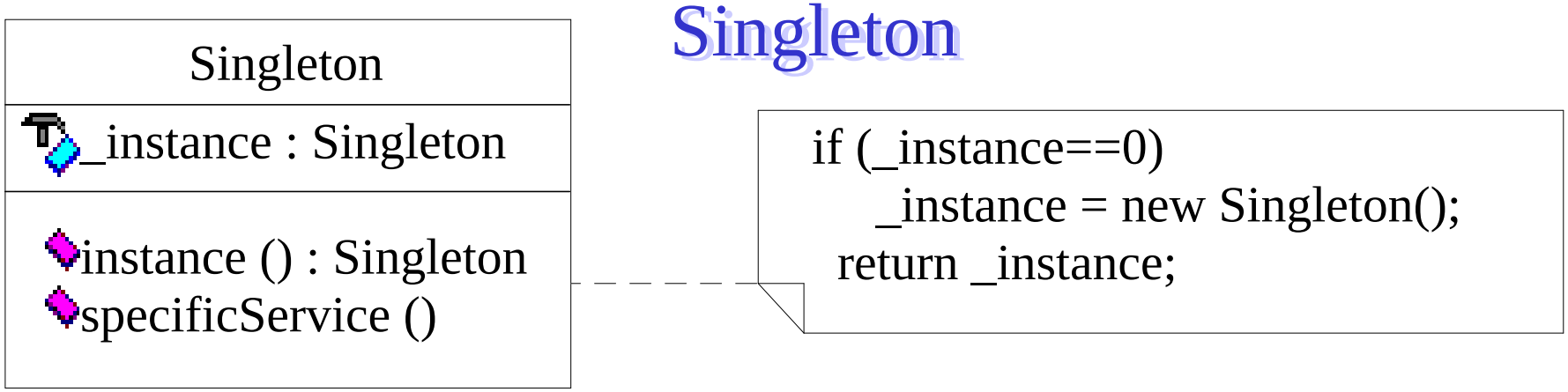
# Factory




I client possono richiedere la creazione di un prodotto senza dipendervi.

La **`Factory`** dipende dai prodotti concreti, mentre i client dipendono solo **`AbstractProduct`**.

# Singleton




```
user_code()
{
  Singleton::instance()->specificService(...);
}
```

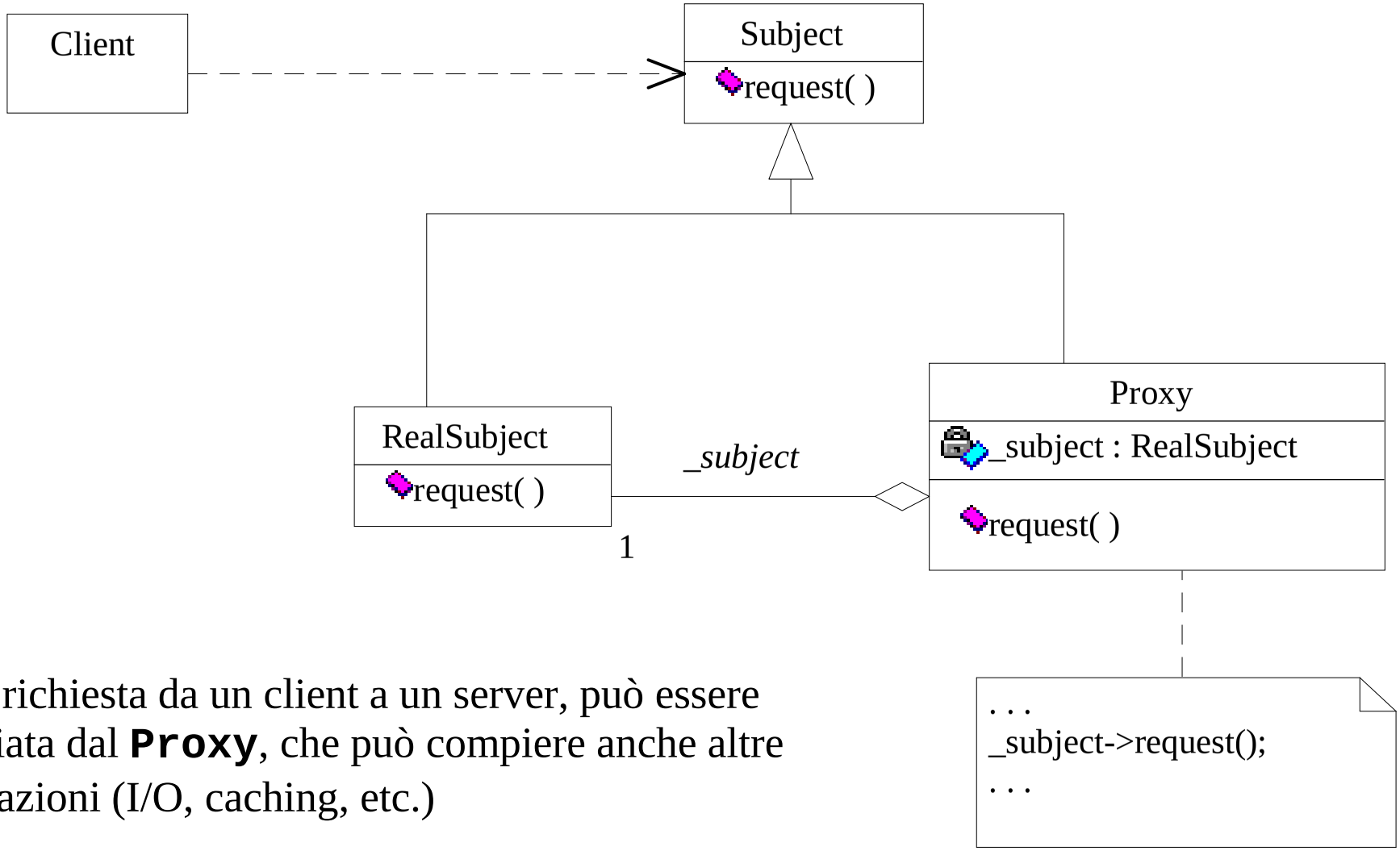
Il Singleton pattern piò essere usato ogni volta che una classe deve essere instanziata una sola volta, e viene usata da diversi oggetti.

Per evitare istanziazione accidentale, il constructor deve essere privato.

Più istanze, ma in numero ben determinato, possono esistere (*multiton*)

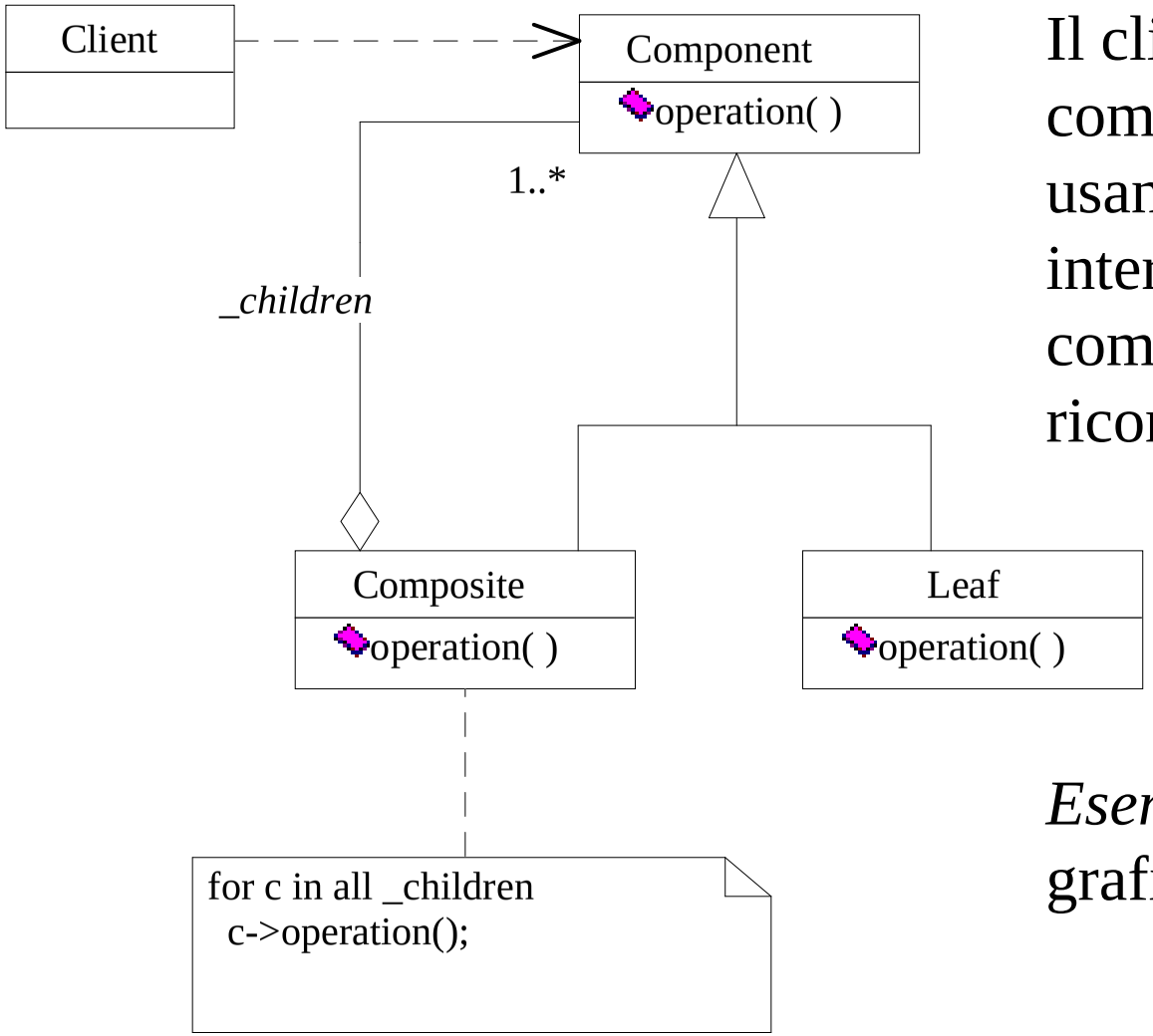
Siccome vengono usate funzioni statiche, l'ereditarietà non può essere applicata.

# Proxy




Una richiesta da un client a un server, può essere mediata dal **`Proxy`**, che può compiere anche altre operazioni (I/O, caching, etc.)

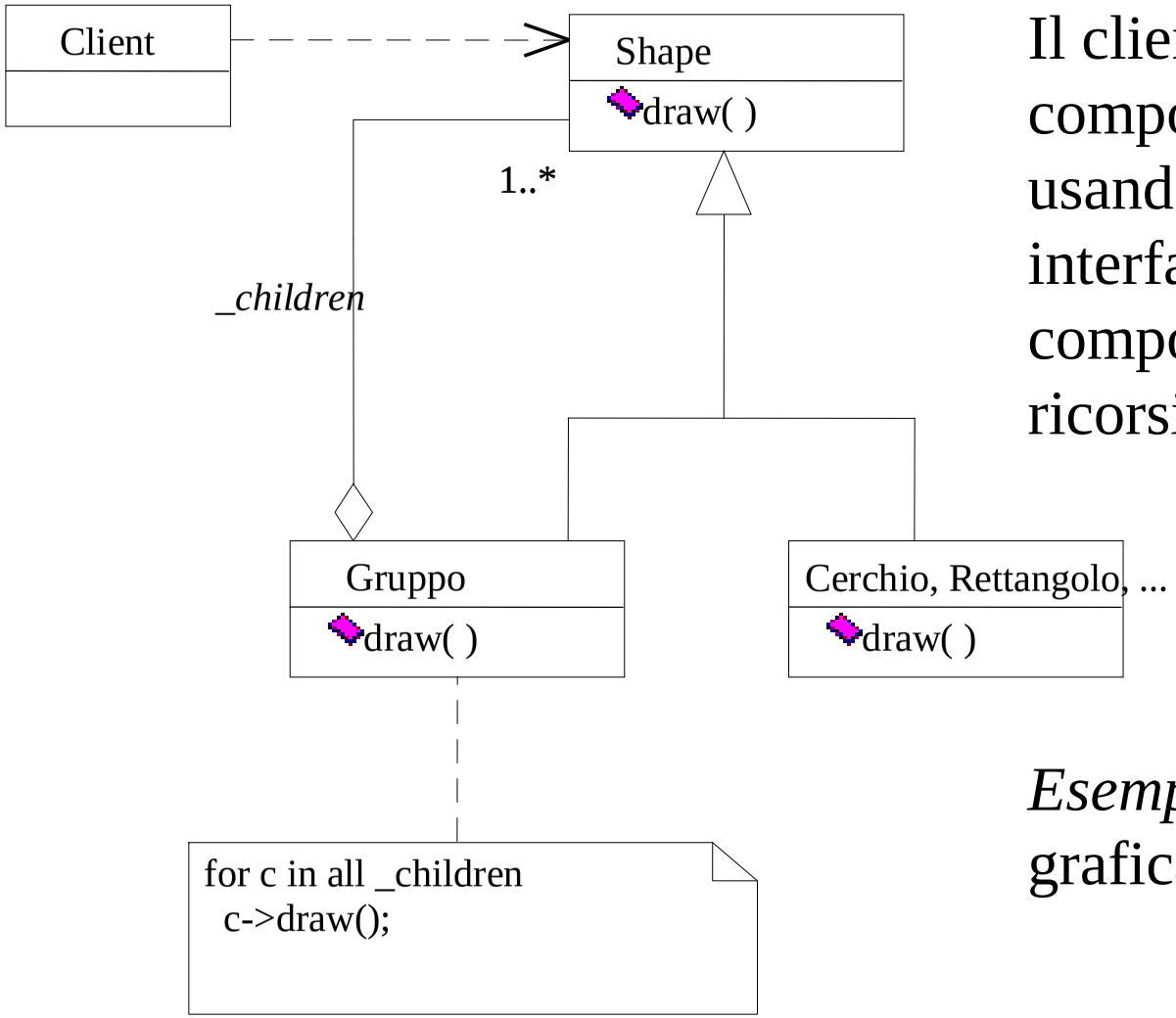
# Composite




Il client può trattare componenti e compositi usando la stessa interfaccia. La composizione può essere ricorsiva.

*Esempio*: programmi di grafica vettoriale

# Composite Shape



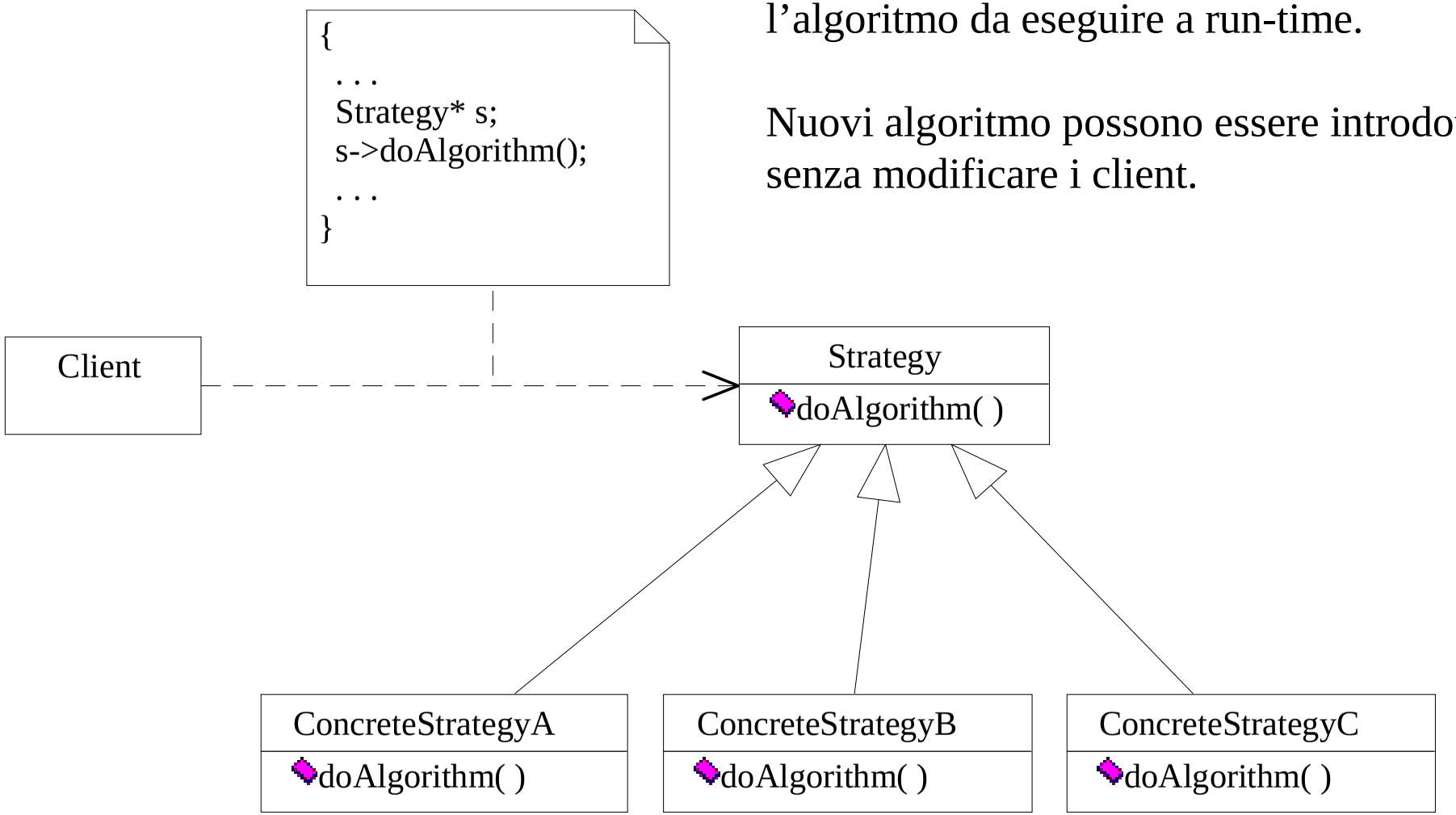

Il client può trattare componenti e compositi usando la stessa interfaccia. La composizione può essere ricorsiva.

*Esempio*: programmi di grafica vettoriale

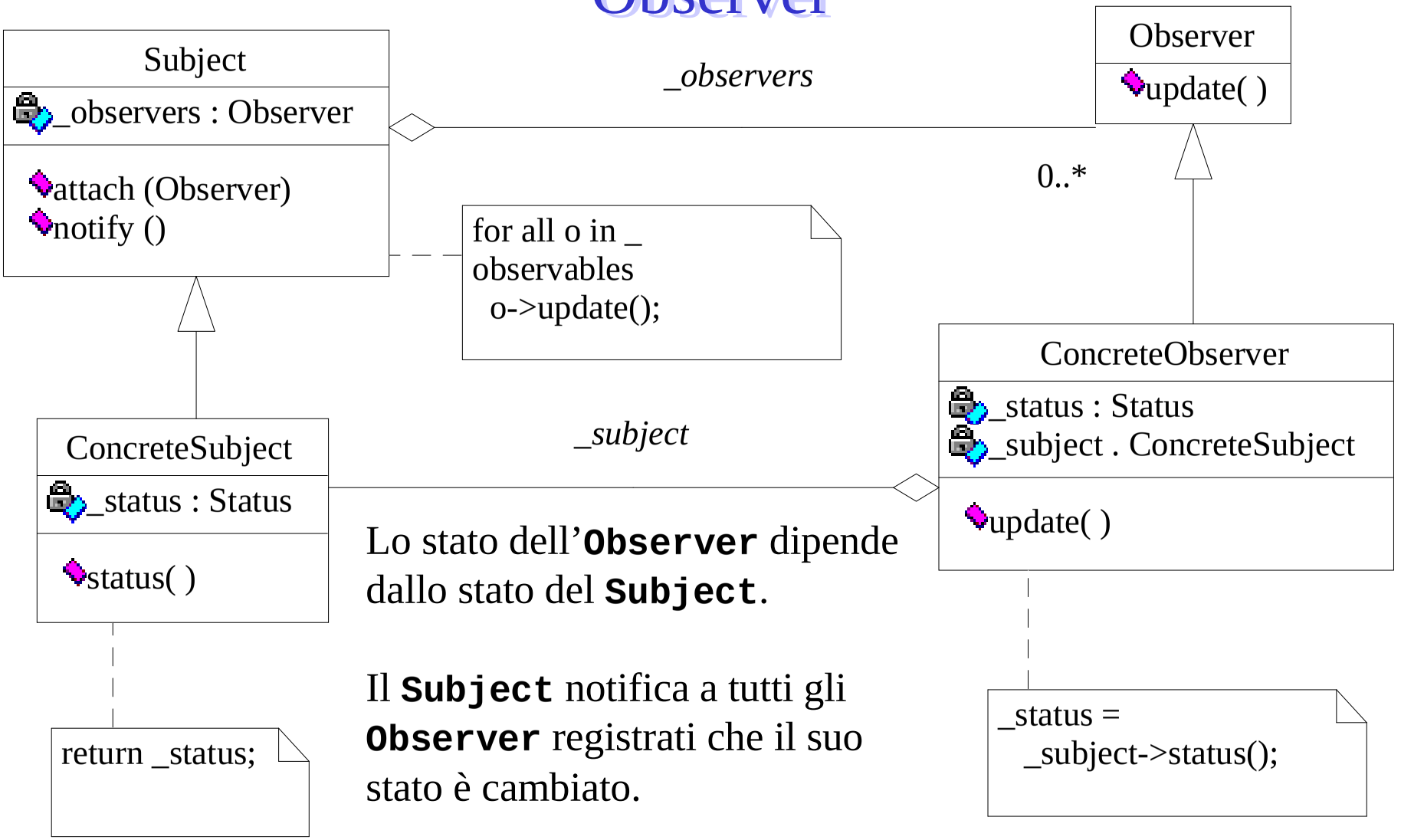
# Strategy

Il pattern Strategy permette di scegliere l'algoritmo da eseguire a run-time.

Nuovi algoritmo possono essere introdotti senza modificare i client.



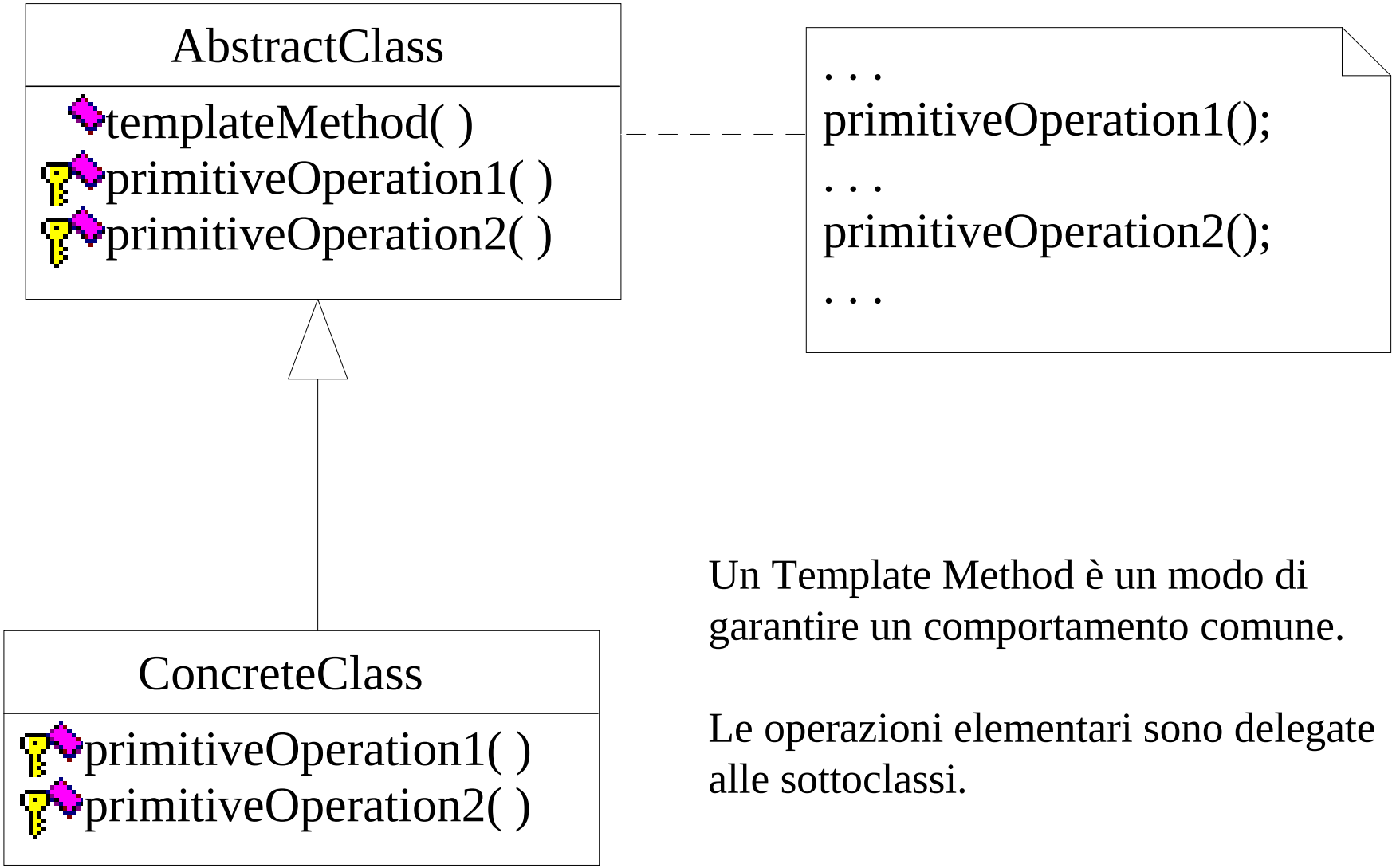
# Observer




Lo stato dell'**`Observer`** dipende dallo stato del **`Subject`**.

Il **`Subject`** notifica a tutti gli **`Observer`** registrati che il suo stato è cambiato.

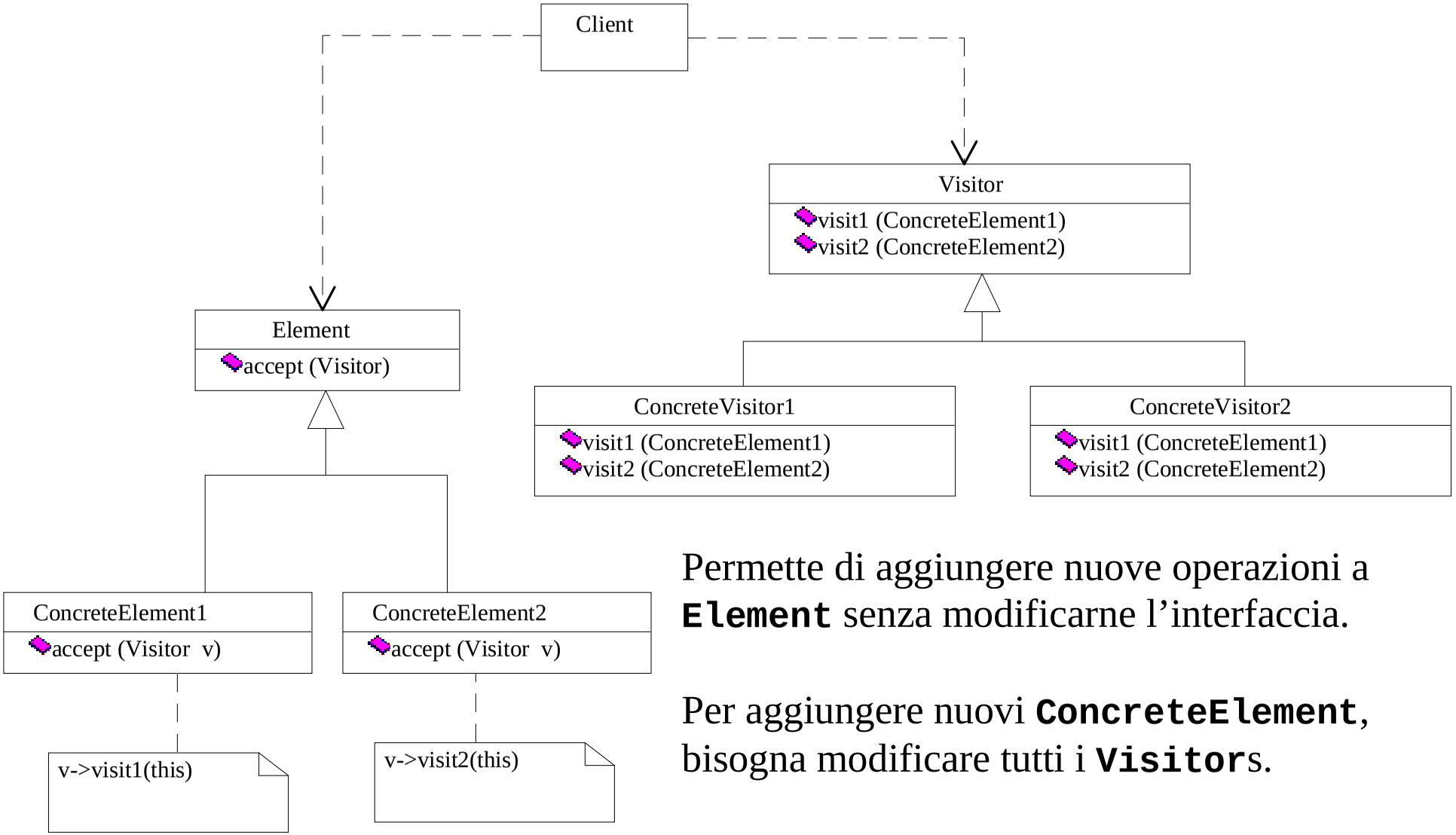
# Template Method




Un Template Method è un modo di garantire un comportamento comune.

Le operazioni elementari sono delegate alle sottoclassi.

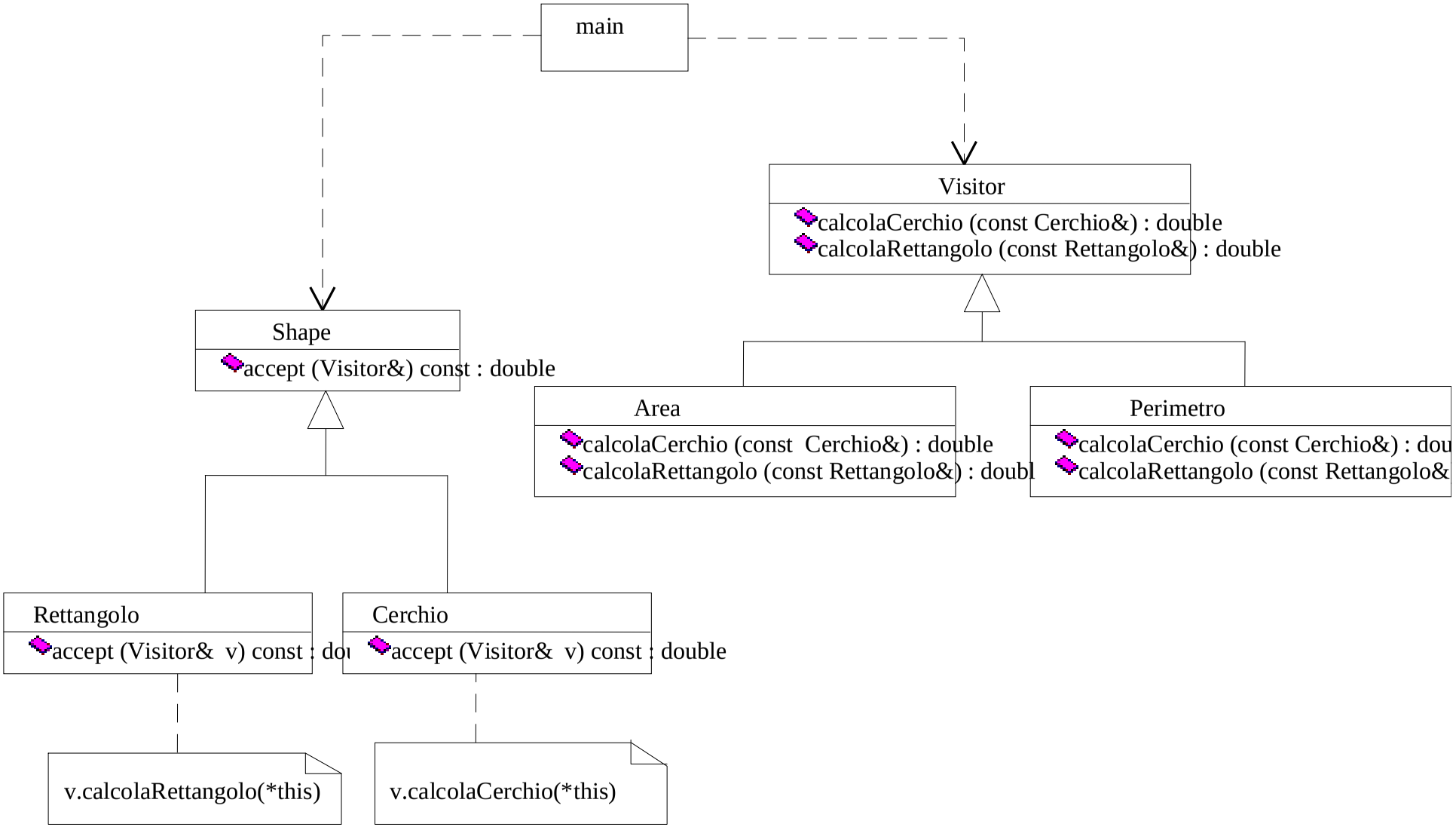
# Visitor




Permette di aggiungere nuove operazioni a **`Element`** senza modificarne l'interfaccia.

Per aggiungere nuovi **`ConcreteElement`**, bisogna modificare tutti i **`Visitor`**s.

# Visitor


main
Visitor
calcolaCerchio (const Cerchio&) : double
calcolaRettangolo (const Rettangolo&) : double
Shape
accept (Visitor&) const : double
Area
calcolaCerchio (const Cerchio&) : double
calcolaRettangolo (const Rettangolo&) : doubl
Perimetro
calcolaCerchio (const Cerchio&) : dou
calcolaRettangolo (const Rettangolo&
Rettangolo
accept (Visitor& v) const : dou
Cerchio
accept (Visitor& v) const : double
v.calcolaRettangolo(*this)
v.calcolaCerchio(*this)