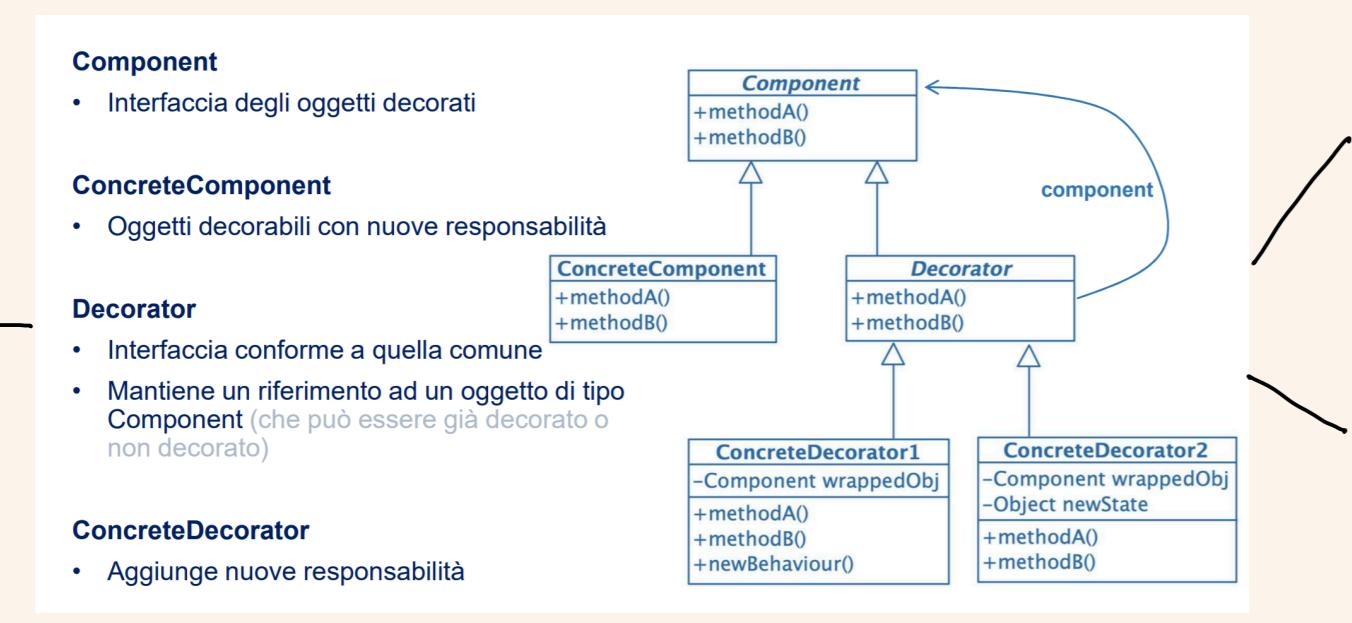


## Design Pattern

Pattern Decorator — aggiunge — dinamicamente — responsabilità — oggetto  
alternativa — flessibile — ereditarietà



ereditarietà — usata — solo — type matching  
comportamento — ottenuto — Composizione — non — estensibile  
permette — combinazione — decorator — runtime

Pro — maggiore — flessibilità — ereditarietà  
più — semplice — ereditarietà — multiple  
favorisce — aggiunta — incrementale — feature

non — supportato — da — ereditarietà

Contro — se — complessi — costosi;

possono — essere — composti — molti — oggetti → debug difficile  
differenti — da — componenti — componente ≠ componente装饰者 → non affidarsi a identità oggetto stesso

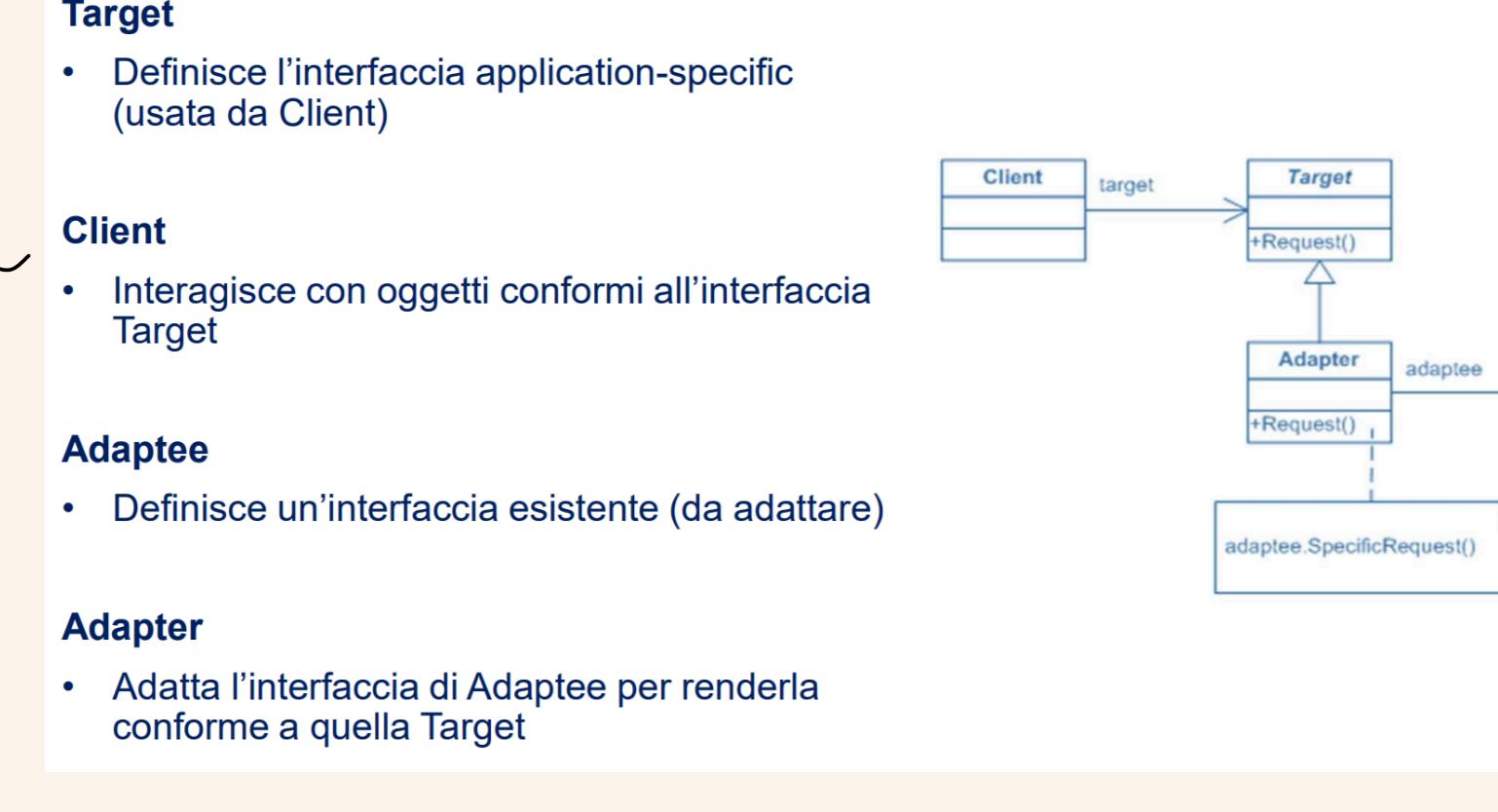
```
// The good way
Beverage b = new DarkRoast();
b = new Mocha(b);
b = new Mocha(b);
System.out.println(b.getDescription() + "$" + b.cost());

// The poor way
Beverage b = new DarkRoast();
b = new Mocha(b);
b = new Mocha(b);
Whip b2 = new Whip(b);
System.out.println(b2.getDescription() + "$" + b2.cost());
```

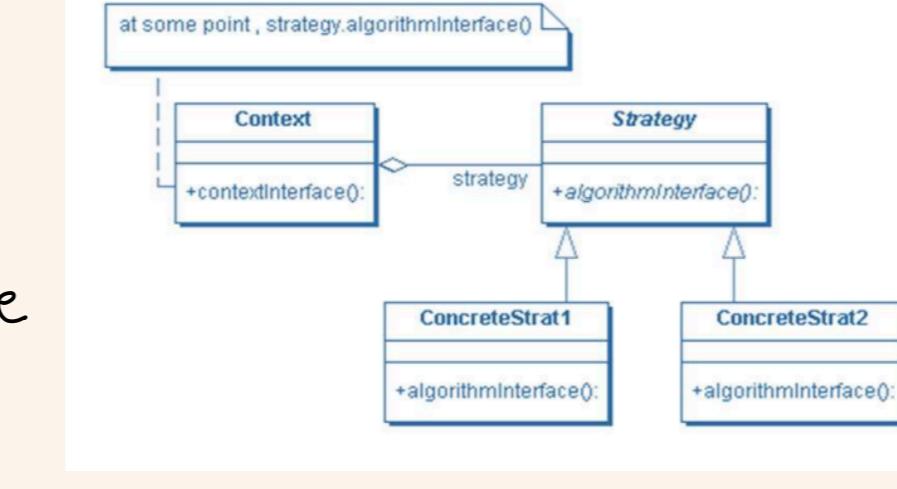
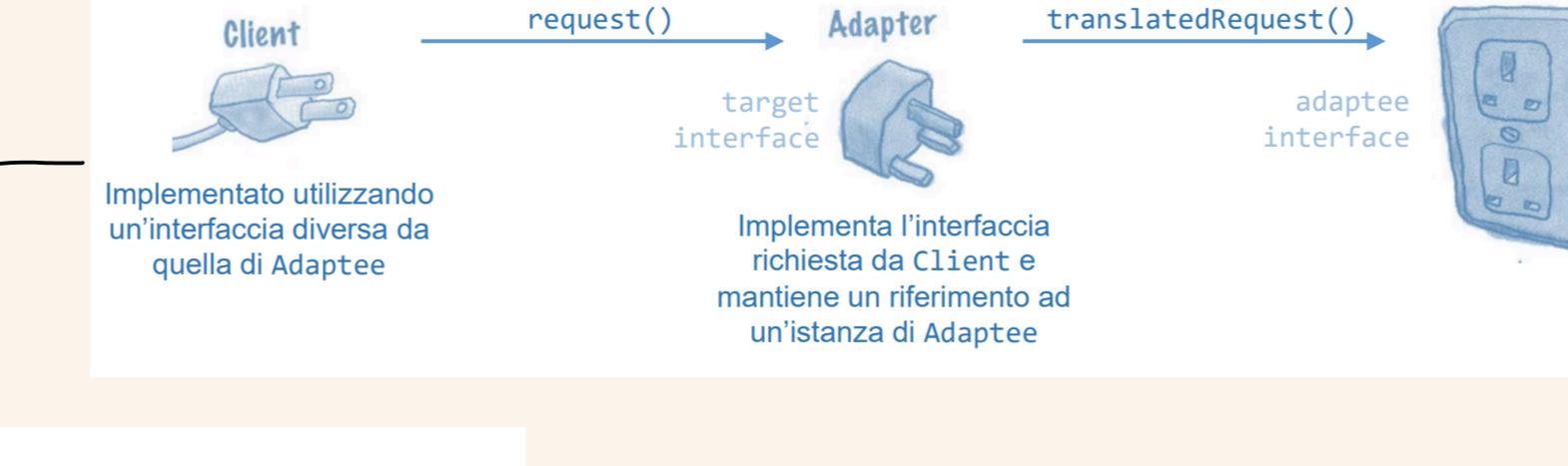
Implementation —

- L'interfaccia di un oggetto decoratore deve essere conforme all'interfaccia del componente che decora
- Se è necessaria una sola «decorazione», non serve definire la classe astratta Decorator (le cui responsabilità possono essere fuse con quelle di ConcreteDecorator)
- Mantenere leggere le classi astratte componenti che sono estese da componenti e decoratori Devono definire solo l'interfaccia comune (e nessuna altra funzione)
  - Se diventano complesse, potrebbero rendere il decoratore troppo costoso da usare
- Le classi decoratore dovrebbero funzionare come «estatico» sopra un oggetto (se gli oggetti devono cambiare internamente, meglio usare lo Strategy pattern)

Pattern Adapter — converte — interfaccia — classe → interfaccia attesa dal cliente  
consente — integrazione — classi — estremamente — non integrabili (interfaccia incompatibili)



target e Adaptee — esistono — prima di — Adapter (sistemi legacy)  
binding — Adapter — Adaptee — realizzato — delega  
ereditarietà — meccanismo — specifica — interfaccia — della — classe Adapter



Pattern Facade — pattern GoF — realizza — adattatore — inoltre — simultaneamente — richieste — più adaptree

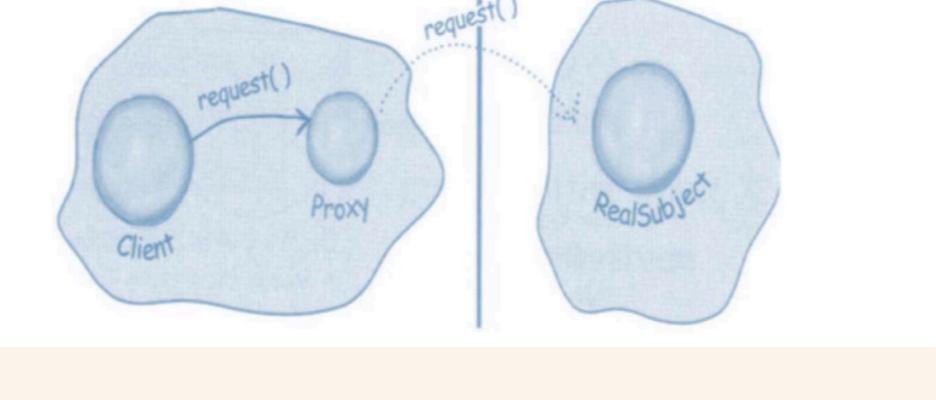
Pattern Proxy — simile — Pattern Adapter — diversi — Proxy Pattern — proxy — oggetto — stessa — interfaccia  
proxy — prov — eseguire — pre/post processing

Proxy — fornisce — surrogato — altro — oggetto — controllarne — accesso (segnaposto)

### ESEMPIO

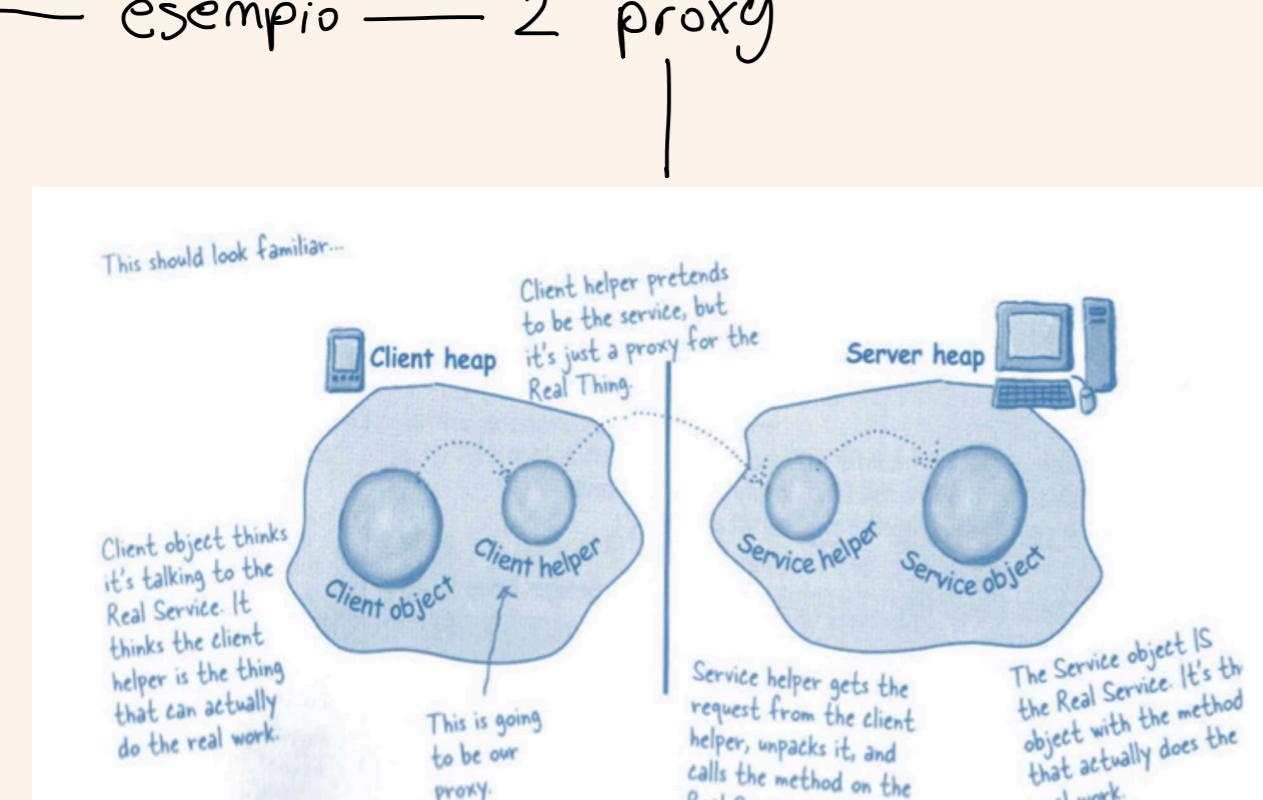
Supponiamo di voler monitorare un servizio GumballMachine

- Possiamo farlo attraverso un **remote proxy** (ovvero mediante una rappresentazione «locale» di un oggetto «remoto»)
- Il client effettua una richiesta al proxy
  - La richiesta viene inviata (attraverso la rete) all'oggetto remoto
  - Il risultato è restituito al proxy
  - Il proxy inoltra il risultato al client



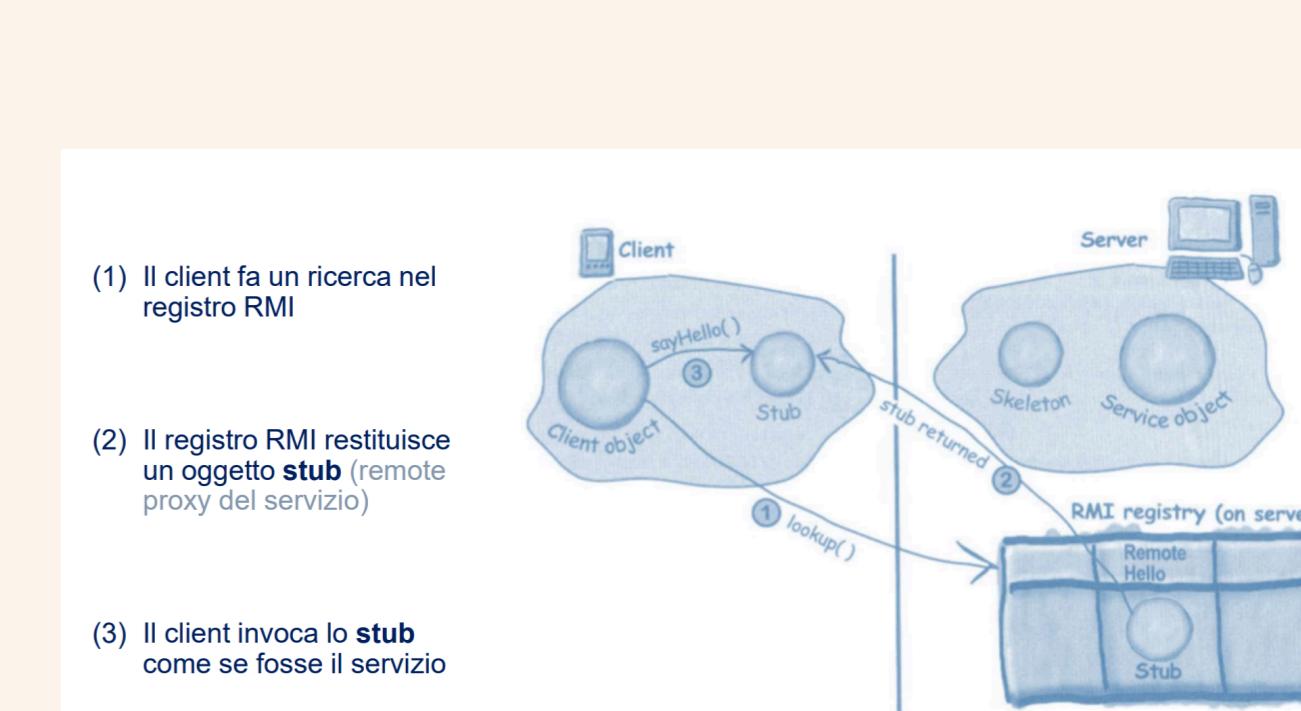
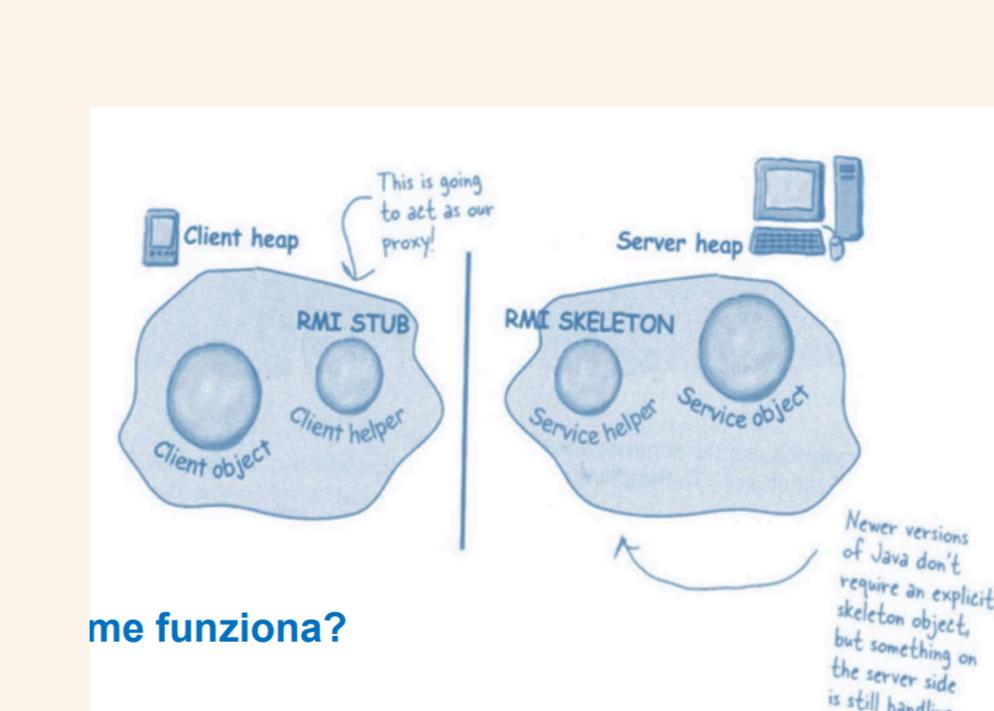
diversi tipi:  
Remote Proxy — permette — accesso — oggetto — remoto  
Protection Proxy — implements — controllo — accessi  
Cache Proxy — mantiene — copie — richieste/risposte  
Synchronization Proxy — gestisce — accessi — concorrenti — servizio  
Virtual Proxy — comporta — come — originale — finché — viene — costruito — poi — pesta — richieste — originale  
utile — ritardare — creazione — oggetto — costoso  
creato — solo — quando — necessario

esempio — 2 proxy



client helper — agisce — stub  
service helper — agisce — skeleton

Simile — Java RMI



esempio — Virtual Proxy

