Il pattern State

a cura di **Angelo Furfaro**
da “Design Patterns”, Gamma et al.

Dipartimento di
Ingegneria Informatica, Elettronica, Modellistica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: a.furfaro@unical.it
Web: http://angelo.furfaro.dimes.unical.it

State

Classificazione

- Scopo: comportamentale
- Raggio d'azione: basato su oggetti

Scopo

- Permette ad un oggetto di cambiare il suo comportamento al cambiare del suo stato interno.
- L'oggetto si comporterà *come* se avesse cambiato la sua classe.

Altri nomi

Object for States

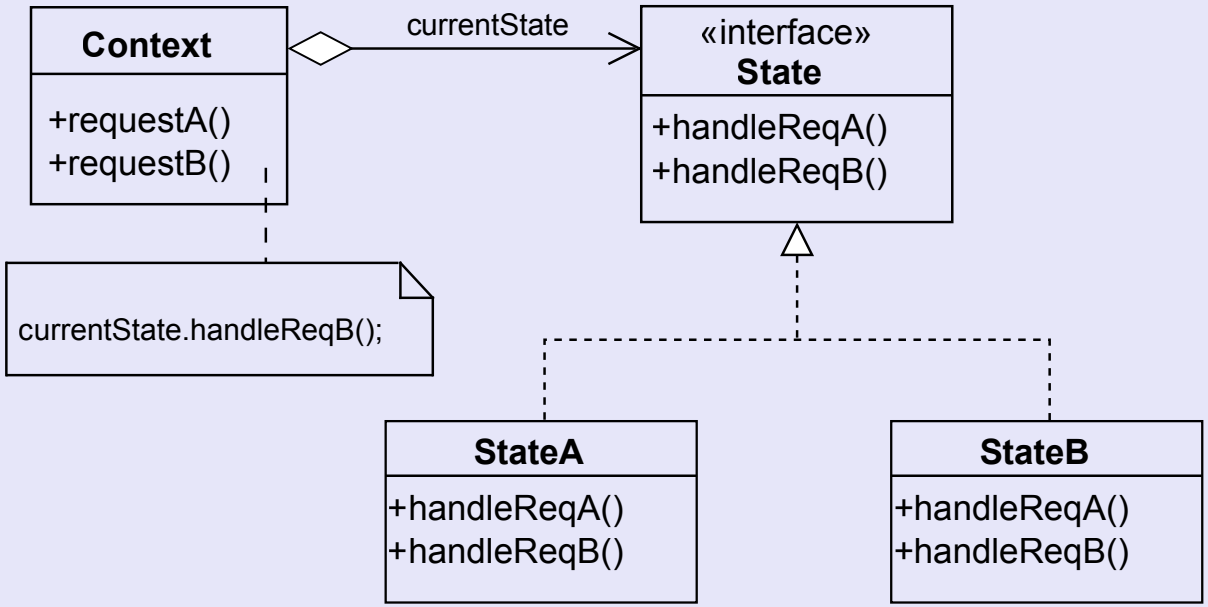
Motivazione

-

Applicabilità

- Il comportamento di un oggetto dipende dal proprio stato e deve cambiare durante l'esecuzione in funzione dello stato corrente.
- Esistono metodi con grandi blocchi di codice per scelte condizionali il cui esito dipende dallo stato dell'oggetto. Di solito lo stato è rappresentato in questi casi da costanti. Il pattern State confina ogni ramo condizionale in una classe separata.

Struttura



Partecipanti

- **Context**: definisce l'interfaccia utilizzata dai client. Mantiene un riferimento ad un'istanza di una classe che implementa l'interfaccia `State` e che rappresenta lo stato corrente.
- **State**: Definisce un'interfaccia che incapsula il comportamento associato ad uno stato particolare di `Context`.
- **ConcreteState**: Ogni classe che occupa questo ruolo definisce un particolare comportamento associato ad uno stato di `Context`.

Conseguenze

- Localizza il comportamento specifico di uno stato e suddivide il comportamento in funzione dello stato:
  - Tutta la logica specifica di uno stato è confinata in un singolo oggetto.
  - Definendo sottoclassi di `State` è possibile aggiungere nuovi statoi e nuove transizioni.
  - Aumenta il numero di classi, ma ciò è auspicabile per evitare grossi blocchi di scelte condizionali.
- Rende esplicite le transizioni di stato:
  - L'introduzione di oggetti diversi per rappresentare stati diversi rende le transizioni tra stati esplicite.
  - Dal punto di vista di `Context` le transizioni sono atomiche: si modifica il riferimento all'oggetto `State` e non un insieme di attributi.
- Gli oggetti `State` possono essere condivisi:
  - Se gli oggetti `State` non hanno varaibili di istanza, allora `Context` diversi potranno condividere le stesse istanze di oggetti `State`. Ciò può essere fatto ricorrendo al pattern *Flyweight*.

Pattern Correlati

Flyweight

Flyweight può essere utilizzato per condividere gli oggetti `State`.

Singleton

Spesso gli oggetti `State` sono dei Singleton.