Design pattern 2

Introduzione

Obiettivi:

- Applicare propriamente i pattern del modello GoF

Questa sezione rappresenta il naturale conseguimento del documento *Design Pattern 1*, in cui sono elencati ulteriori pattern *comportamentali*, *creazionali* e *strutturali* del *catologo GoF*.

State Pattern

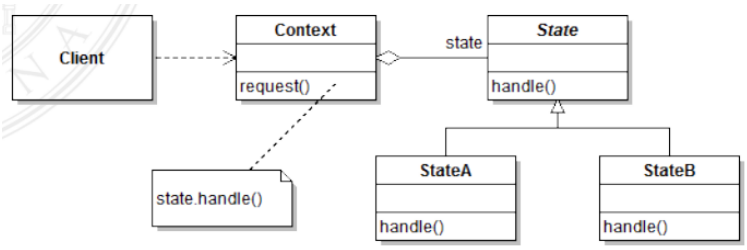
Problema
Come intercambiare il comportamento di un oggetto dipendente da uno stato tramite una soluzione di alta qualità?

Soluzione
Definire un meccanismo che permetta ad un oggetto di variare il proprio comportamento quando lo stato interno dell'oggetto varia.

Il pattern **State** è una soluzione applicata alla necessità di creare un insieme di comportamenti caratteristici strettamente correlati ad uno stato, che possa acquisire l'oggetto della classe in questione. In relazione all'adozione di un meccanismo di composizione simile, sono adottati sei passi principali:

- Creare un layer astratto, quale un'interfaccia, che definisca l'intero contesto a cui una classe possa richiedere di variare il proprio comportamento
- Creare una classe astratta da cui deriveranno le classi concrete, ossia che implementano i *behavior* specifici
- Rappresentare i differenti stati acquisibili dal sistema software modellato come sottoclassi derivanti dalla classe astratta principale
- Implementare il meccanismo comportamentale specifico per ogni singola sottoclasse derivata
- Mantenere un riferimento all'interno dell'interfaccia affinchè sia possibile variare il comportamento in base allo stato corrente dell'istanza
- Concludendo, per variare lo stato del sistema software, modificando il comportamento associato, semplicemente verrà cambiato il riferimento dello *stato corrente*

Caso di studio
Affinchè sia possibile comprendere quando e come applicare al meglio il pattern *State*, di seguito è proposto un esempio grafico che possa raffigurare al meglio quanto detto.



Come da raffigurazione si notano i passaggi elencati precedentemente, in cui la classe *Context* rappresenta il livello di astrazione necessario affinchè sia possibile intercambiare comportamenti di oggetti durante il run time.
Context a sua volta delega alla classe astratta *State* la responsabilità di implementare il comportamento specifico, in cui in questo caso avviene per uno dei due *behavior* esternalizzati, *StateA* e *StateB*; è bene sottolineare la sottile differenza che vige dal pattern *Strategy*, dove nel meccanismo di delega introdotto la scelta del *behavior* è imposta dall'utente finale, si ricorda il processo modellato, in cui è definita l'architettura generale dell'algoritmo affinchè siano classi derivate a specificarne l'implementazione di step specifici, mentre mediante *State* sarà l'interfaccia ad imporre quale metodo debba essere considerato affinchè sia garantito il cambiamento dello stato corrente, il quale gioca un ruolo fondamentale poichè permette intercambiabilità del processo comportamentale dell'istanza in questione.

Infine si considera la casistica in cui debbano essere introdotte nuove funzionalità. Dato che si considera lo stretto legame posto tra *comportamento-stato*, ogni volta che si debba implementare un nuovo *behavior* sarà creata una nuova classe derivata che specificherà la logica logaritmica del metodo, ovviando a *design smells*, come *needless repetition*, e violazioni dei principi *SOLID*.

Factory Method Pattern

Problema
...

Soluzione
...

Caso di studio
...