# Design Pattern

venerdì 22 novembre 2024 16:44

Design Pattern — pattern descrivono problema e nucleo soluzione riutilizzabile

GOF — classificazione pattern — usa Object Modeling Technique

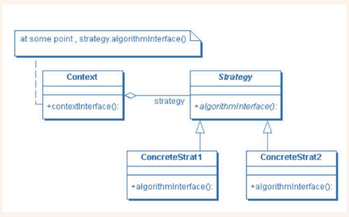
- Creazionali — riguardano creazione oggetti
- Strutturali — composizione (classi / oggetti)
- Comportamentali — interazioni tra (classi / oggetti) — suddivisione responsabilità

## Strategy Design Pattern

definire famiglia algoritmi — incapsularli separatamente — renderli intercambiabili

fornisce varianti algoritmo/comportamento — classi separate

metodo uniforme — accesso classi




**Strategy** — definisce interfaccia comune algoritmi

**ConcreteStrategy** — implementa algoritmo

**Context** — contiene riferimento oggetto Strategy

può definire interfaccia

consente strategia accedere dati

### Quando si usa?

più classi differiscono solo comportamento

servono più varianti algoritmo

evitare esporre strutture complesse — specifiche per algoritmo

### Osservazioni

consente al prezzo di:

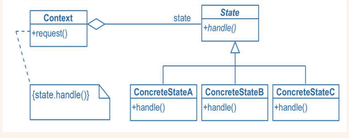
- definire comportamenti diversi (estensione Context)
  - eliminare costrutti condizionali
    - scegliere implementazioni diverse per stesso task
- incrementa numero oggetti implementazione
  - rispettare stessa interfaccia varie implementazioni

## State Design Pattern

pattern comportamentale — consente oggetto alterare comportamento quando cambia stato

**Context** — definisce interfaccia interesse client

mantiene istanza **ConcreteState** definisce stato corrente




**State** — incapsula comportamento associato stato contesto

può essere classe concreta

**ConcreteState** — sottoclassi stato implementano comportamento stato

### Quando si usa?

comportamento oggetto dipende stato

operazioni hanno dichiarazioni condizionali complesse

### implementazione in 6 passi

1. Identificare/creare una **classe** che funga da «**macchina a stati**» dal punto di vista del cliente
   - Questa classe è la classe di contesto (**context**)
2. Creare una classe **state** che replichi i metodi dell'interfaccia della macchina a stati
   - Ogni metodo richiede un parametro aggiuntivo: un'istanza della classe di contesto
   - La classe state specifica qualsiasi comportamento utile «predefinito»
3. Creare una **sottoclasse** della classe **state** per ogni stato del dominio
   - Ogni sottoclasse sovrascrive solo i metodi che cambiano con lo stato
4. La classe di **contesto** mantiene lo **stato corrente**
   - Lo stato corrente è un oggetto della classe **state**
5. Le richieste dei client sono **delegate** dalla classe di contesto allo **stato corrente**
   - Passate all'oggetto della classe state
   - Si passa anche il puntatore `this` all'oggetto di contesto
6. I **metodi della classe state modificano lo stato corrente**
   - Modifica dell'oggetto di contesto passato

- State «localizza» il comportamento che dipende dallo stato
  - Il comportamento è suddiviso tra i possibili stati
  - Le transizioni di stato sono esplicite
- Tutti i comportamenti specifici di uno stato sono memorizzati in una classe
  - L'alternativa è costituita da «valanga» di casi (condizionali) alternativi
  - Può produrre un gran numero di classi, ma è meglio dell'alternativa
- Lo stato corrente è memorizzato in un'unica posizione
- Gli oggetti della classe state possono essere condivisi
  - Quando non hanno variabili di istanza
- La classe **state** può implementare parte del comportamento
  - Se c'è del comportamento comune o di default

## State vs Strategy

similitudini basati composizione con **delega**

differenza intento

- Strategy — incapsula algoritmo
- State — incapsula (comportamento / transizioni stato)