Design Patterns

- I principi SOLID, Robert C. Martin
  - Agile Software Development, Principles, Patterns, and Practices, 2002
  - Clean Architecture: A Craftsman's Guide to Software Structure and Design, 2017
- Un design pattern è una soluzione verificata a un problema comune
  - Progettazione più flessibile e modulare, documentazione del codice più intuitiva
  - Trasposizione di un’idea nata in ambito architetturale (edile / urbanista)
    - A pattern language di Christopher Alexander, 1977
    - Using Pattern Languages for Object-Oriented Programs
      - ACM OOPSLA, Technical Report No. CR-87-43, Kent Beck, Ward Cunningham, 1987
  - Design Patterns: Elements of Reusable Object-Oriented Software
    - Erich Gamma et al. (GoF: Gang of Four), 1994

Principi di design SOLID

- Single Responsibility
  - Deriva dalla “Separation of Concern” (Edsger W. Dijkstra) dal punto di vista dell’attore
- Open/Closed
  - Chiusura al cambiamento (se non per il bug fixing), apertura all’estensione (Bertrand Meyer)
- Liskov Substitution
  - Al posto di un oggetto di una classe, un oggetto di una sua classe derivata
  - Vedi anche Design By Contract (Bertrand Meyer)
- Interface Segregation
  - Non bisognerebbe dipendere da metodi che non si usano
  - Una classe concreta implementi più interfacce, l’oggetto istanziato sia gestito via l’interfaccia di interesse
- Dependency Inversion
  - I moduli di alto livello non dovrebbero dipendere da quelli di basso livello
  - I dettagli implementativi dovrebbero essere definiti in termini quanto più possibili astratti
  - Inversion of Control: enfasi sulla divisione dei compiti e debolezza della connessione tra i moduli

Definizione di pattern

- Nome
  - Descrive il pattern e la sua soluzione in un paio di parole
- Problema
  - Contesto e ragioni per applicare il pattern
- Soluzione
  - Elementi del design, relazioni, responsabilità e collaborazioni
- Conseguenze
  - Risultato, costi e benefici, impatto sulla flessibilità, estensibilità, portabilità del sistema
  - Possibili alternative

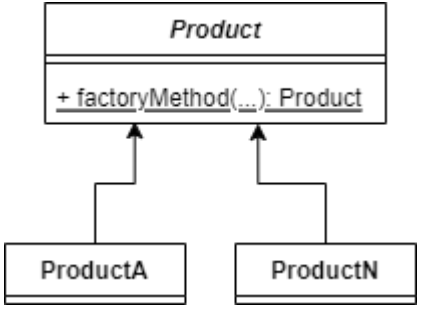
Classificazione dei pattern

- Scopo
  - Creazionali
    - Creazione / clonazione di oggetti
  - Strutturali
    - Generalizzazione
    - Decomposizione
  - Comportamentali
    - Interazione tra oggetti o classi
    - Flusso di controllo
- Raggio d’azione
  - Classe (statico)
    - Ereditarietà
  - Oggetto (dinamico)
    - Associazione
    - Interfacce

Factory Method

pattern creazionale

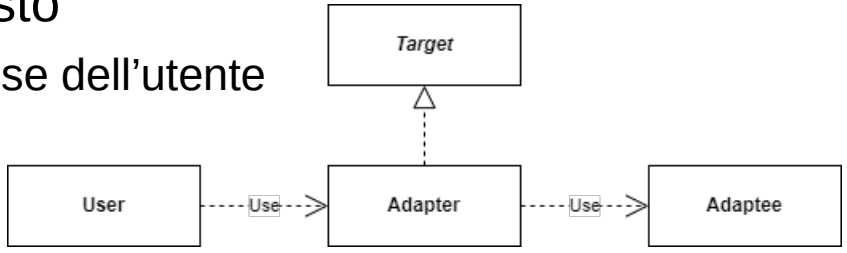
- Data una **gerarchia di classi** che può variare spesso
  - Che ha per base una interfaccia o una classe astratta
- Vogliamo disaccoppiare la creazione di un oggetto dalla sua classe concreta
- Lo scopo è ottenere una maggior flessibilità nella gestione della gerarchia
- Il processo di creazione è delegato a un metodo factory
  - Determina la classe dell'oggetto in base ai parametri passati
  - Regolamenta l'accesso alla gerarchia
    - Può variare all'insaputa dell'utente
- Nell'approccio più leggero basta una singola classe factory
  - Può anche corrispondere alla stessa classe base astratta della gerarchia di riferimento



Adapter

pattern strutturale

- Due classi devono comunicare ma non hanno interfacce compatibili
- L'adapter fa da wrapper per una delle due classi
  - Adattando il flusso di dati a quanto richiesto
    - Offrendo una interfaccia compatibile alle attese dell'utente
    - Usando l'interfaccia nativa del chiamato

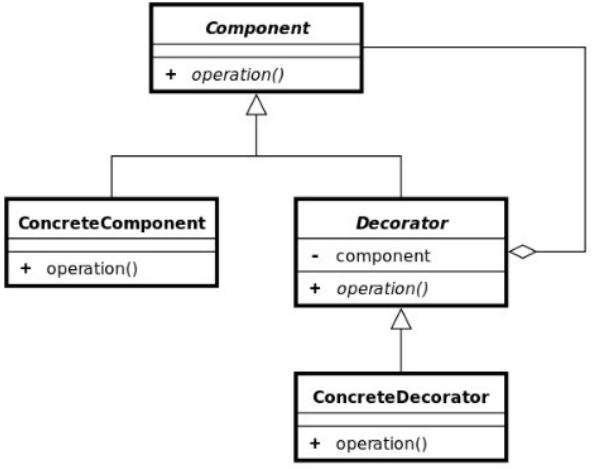


- Approccio alternativo
  - Refactoring di una delle due classi per renderla compatibile all'altra
    - A volte basta che una delle due classi cambi la sua interfaccia
  - È spesso più costoso, rischioso, e può impattare su codice legacy

Decorator

pattern strutturale

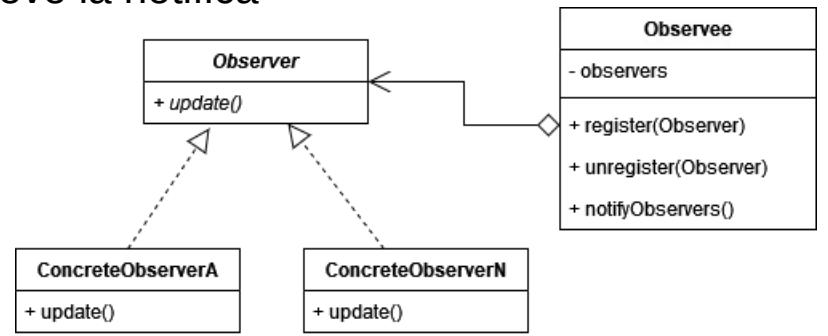
- Abbiamo la necessità di aggiungere funzionalità ad un oggetto, ma stiamo usando un linguaggio staticamente tipizzato
- Modifica il comportamento di un oggetto
  - A runtime
  - Per aggregazione
- Si può creare uno stack aggregato di oggetti
  - Ogni oggetto conosce il successore
  - E gli aggiunge nuove funzionalità



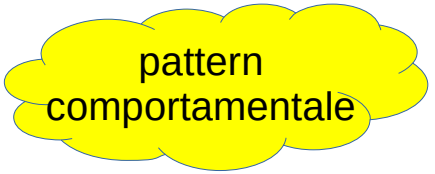
Observer

pattern comportamentale

- Ad alcuni oggetti (osservatori) interessa lo stato di un altro oggetto (osservato)
- Diamo all'osservato la responsabilità di comunicare i cambiamenti
  - È l'osservato che sa quando è necessario notificare gli osservatori
- Gli osservatori implementano un'interfaccia
  - Usata dall'osservato per comunicare con loro
- L'interfaccia definisce il metodo con cui l'osservato notifica i suoi cambiamenti di stato
  - Ogni osservatore può decidere cosa fare quando riceve la notifica
- L'osservato
  - Mantiene un elenco di osservatori
    - Che può essere aggiornato a richiesta
  - Notifica tutti gli osservatori dei cambiamenti



Visitor



- Permette di aggiungere funzioni virtuali a una famiglia di classi senza modificarle
- Le principali componenti nel pattern sono:
  - Visitor: con i metodi virtuali da aggiungere alla famiglia di classi
    - Le classi concrete specificano le operazioni da compiere
  - Element: base della gerarchia che si vuole arricchire
    - Le classi concrete invocano il metodo di visita passando l'oggetto corrente
  - Structure: una collezione di oggetti su cui opera il pattern