# POO - Design Patterns 2

# Structural Patterns

# Structural Patterns

Patern-urile de tip structural ajută la asamblarea obiectelor și claselor în structuri mai mari păstrându-le flexibile și eficiente.

Cei mai folosiți sunt următorii:

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Proxy
- Flyweight

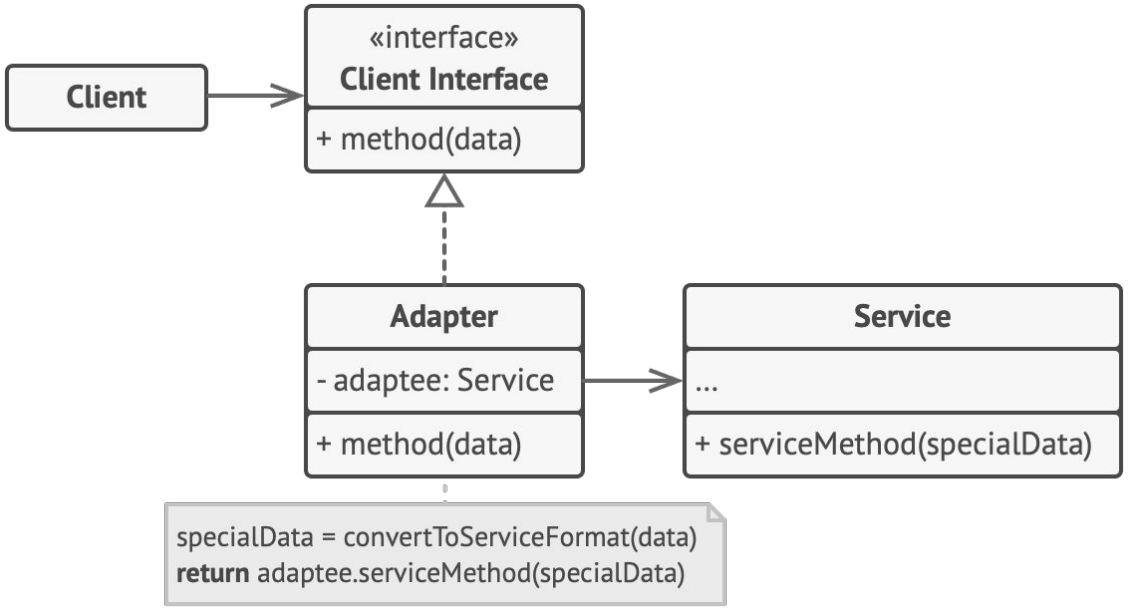
# Adapter

Adapter este folosit pentru a facilita comunicarea dintre clase care nu sunt compatibile datorită interfețelor lor. Astfel se convertește interfața uneia dintre clase astfel încât să poată comunica cu cea existentă.

Adaptorul crează o interfață compatibilă cu un obiect existent.

Prin această interfață, un obiect poate apela metodele adaptorului.

În momentul în care se face o apelare, adaptorul transmite request-ul către cel de-al doilea obiect dar într-un format prelucrat de către adaptor.




Adapter Pattern Implementare

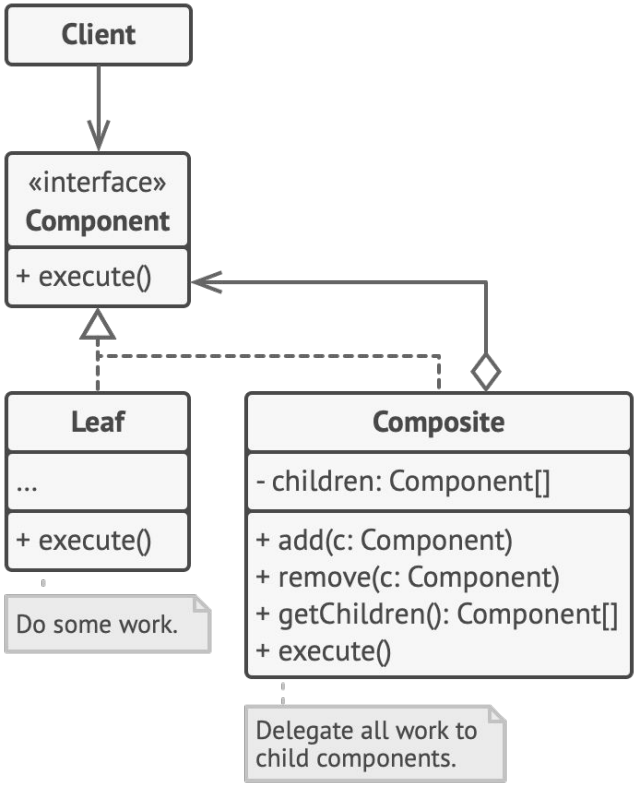
# Composite

Composite este un design structural cu ajutorul căruia putem combina obiecte în structuri de arbore pe care le putem folosi ca și cum ar fi obiecte individuale.

Aplicația trebuie să fie gândită astfel încât aplicația să fie reprezentată sub structură de arbore. Fiecare componentă trebuie să reprezinte un element sau un container. Un container trebuie să aibă atât elemente cât și alte containere.

Se declară o interfață de componente care să conțină metode atât pentru obiecte simple cât și pentru complexe.

Se crează clase frunză (leaf) pentru a reprezenta elemente simple. Se crează o clasă container pentru a reprezenta elemente complexe. Se crează metode pentru adăugarea și eliminarea de elemente copil la container.

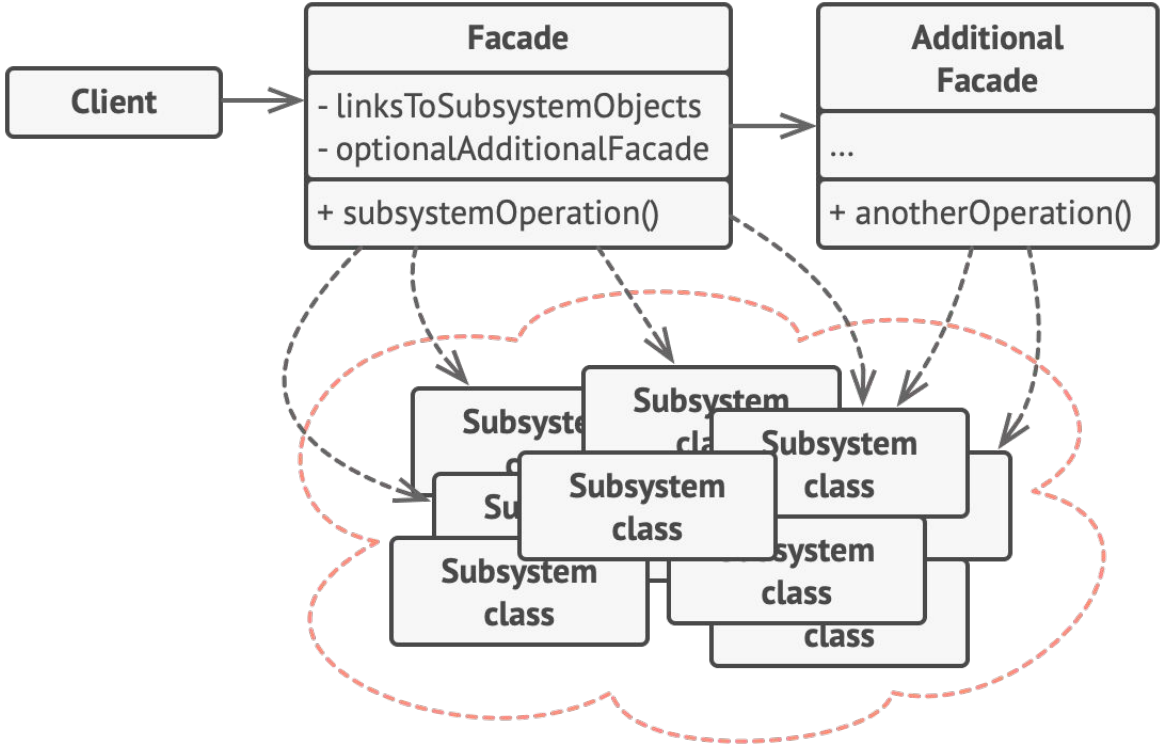



Composite Pattern

# Facade

Facade este un pattern care ascunde complexitatea unui sistem prin implementarea unei interfețe prin care clientul poate accesa sistemul. Această interfață face accesul la subsisteme să fie mai facil.

Se crează o interfață pentru a separa accesul unui cod client de un subsistem de clase, astfel fațada redistribuie apelările către clasele din subsistem.




Facade Pattern

# Behavioral Patterns

# Behavioral Patterns

Pattern-urile de comportament sunt folosiți pentru a identifica comunicații comune dintre obiecte. Astfel se manageriază algoritmii și responsabilitățile dintre obiecte.

Cei mai folosiți sunt următorii:

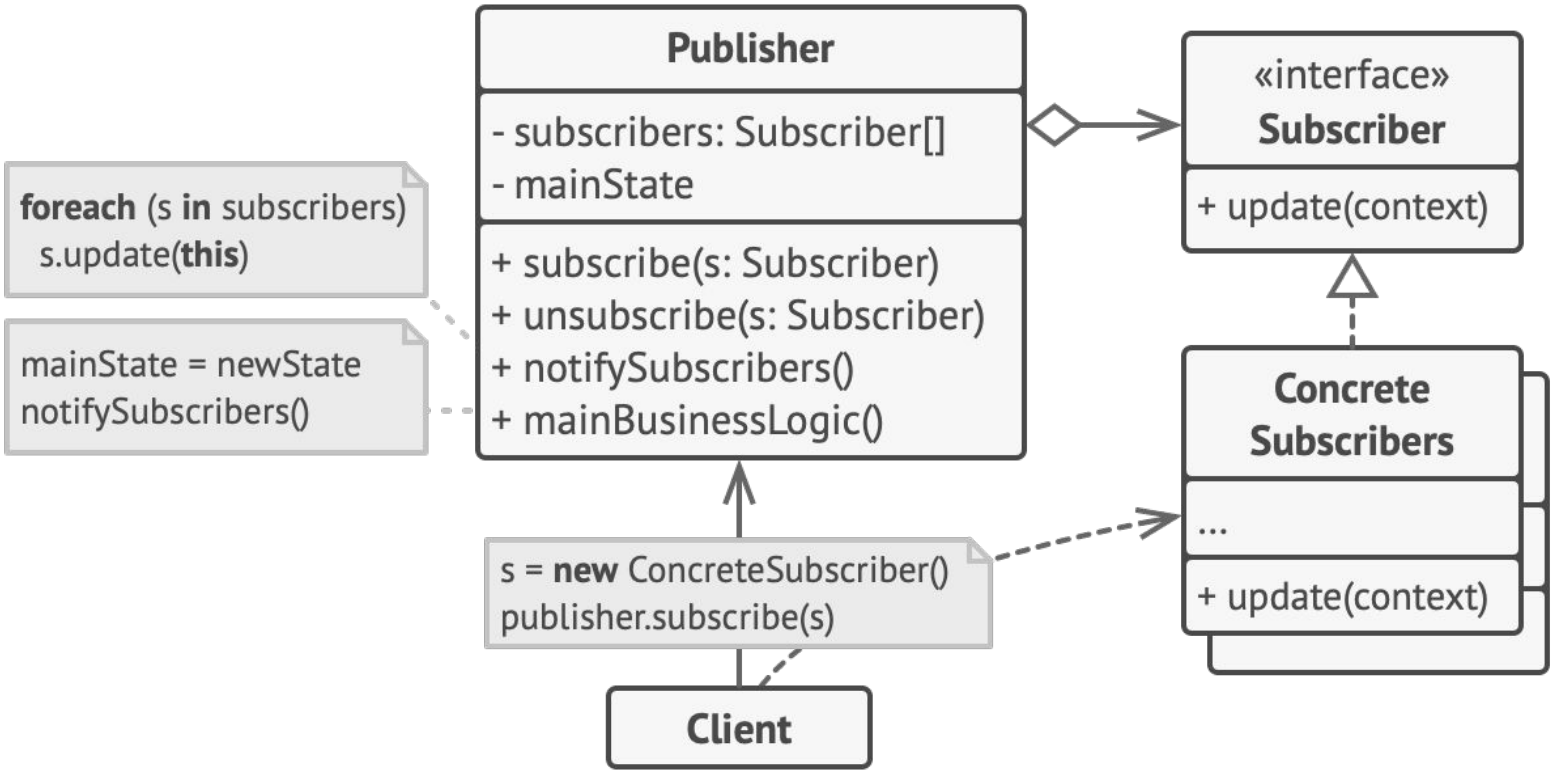
- Observer
- Chain of responsibility
- Visitor
- Iterator
- Mediator
- Strategy
- Template method

# Observer

Observer este un patern prin care se crează un mecanism de subscripție pentru a notifica diferite obiecte despre evenimente care se întâmplă la obiectul observat.

Există un Publisher care trimite notificări către obiectele care fac parte din lista subscriberi. Când Publisher-ul își schimbă starea atunci apelează o metodă de notificare declarată într-o interfață de subscriber, care în general e doar o metodă de update.

Când sunt notificați subscriberii, aceștia primesc parametrii din metoda de update și implementează o logică unică. De multe ori aceștia primesc și o informație contextuală direct de la publisher.



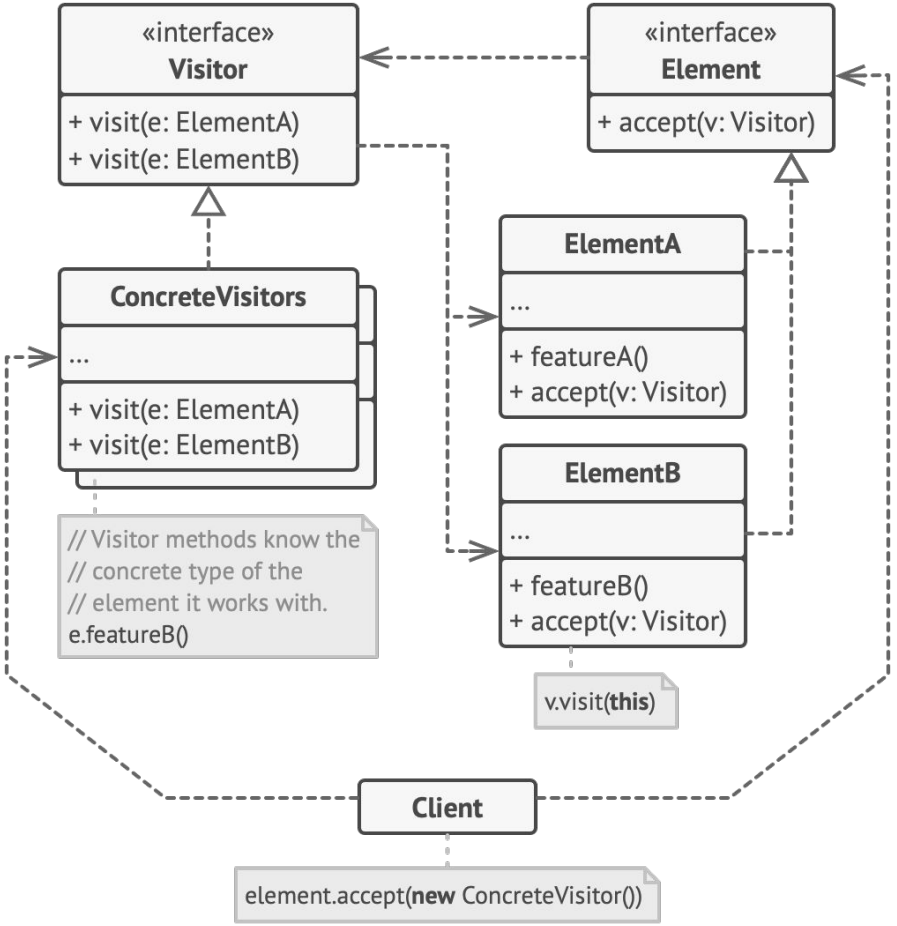

Observer Pattern

# Visitor

Visitor este un pattern prin care se separă algoritmii de obiectele prin care aceștia sunt folosiți.

Se declară o interfață cu un set de metode de vizitare, câte una pentru fiecare clasă care face parte din pattern. Această interfață este implementată de o clasă Concrete Visitor. Această clasă apelează clasele care pot fi vizitate.

De asemenea, clasele care pot fi vizitate au metode de validare și acceptare a vizitatorului, metodele fiind unele implementate de la o interfață de Elemente.

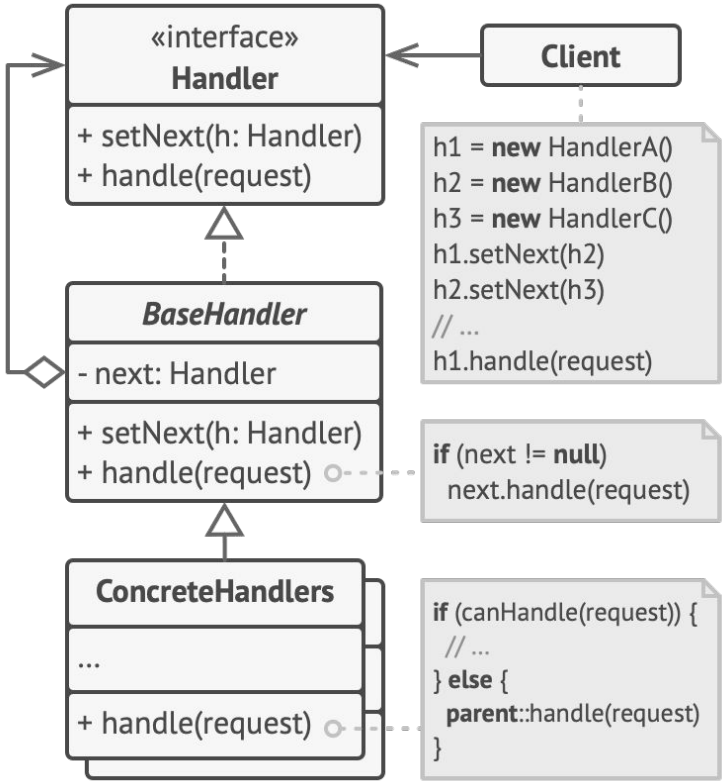



Visitor Pattern

# Chain of Responsibility

Chain of Responsibility este un pattern prin care se procesează request-uri printr-un lanț de decizii. De fiecare dată când un request este primit, fiecare handler decide dacă să proceseze request-ul sau să îl dea mai departe.

Există un handler care declară o interfață comună pentru toți handlers din lanț. Aceasta deobicei are o metodă de a face handle la request-uri și mai poate avea o metodă de a seta următorul handler din lanț.




Chain of Responsibility Pattern

# Summary

# De Reținut

Patern-urile de tip structural ajută la asamblarea obiectelor și claselor în structuri mai mari păstrându-le flexibile și eficiente.

Exemple: Adapter, Bridge, Composite, Decorator, Facade, Proxy, Flyweight

Pattern-urile de comportament sunt folosiți pentru a identifica comunicații comune dintre obiecte. Astfel se manageriază algoritmii și responsabilitățile dintre obiecte.

Exemple: Observer, Chain of responsibility, Visitor, Iterator, Mediator, Strategy, Template method

# Exerciții

# Exerciții

Să se implementeze la alegere 2 pattern din fiecare tip, structural și behavioral (în total 4).

În cod să aveți o zonă comentată în care să prezentați pros and cons pentru fiecare dintre ei. **2p/pattern**