

Metoda FACTORY

Scop

Definește o interfață pentru crearea unui obiect, dar lasă subclasele să decidă ce clasă să instanțieze.

Metoda Factory permite unei clase să defere instanțierea subclaselor

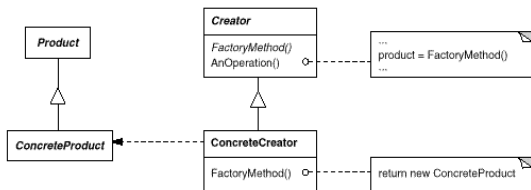
Cunoscut și ca > Constructor Virtual

Aplicabilitate

O clasă nu poate anticipa clasa obiectelor pe care trebuie să le creeze

O clasă dorește ca subclasele sale să specifice obiectele pe care le creează

Clasele delegă responsabilitatea uneia sau mai multor subclase ajutătoare



Sablonul PROTOTYPE

Scop

Specificați tipul obiectelor care se creează folosind o instanță prototipică

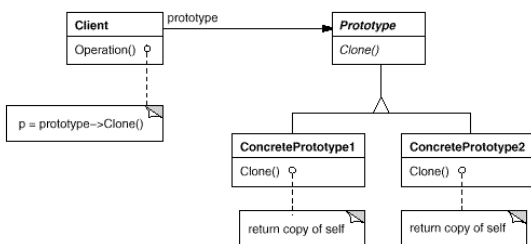
Creați obiecte noi copiind acest prototip

Aplicabilitate

când un sistem trebuie să fie independent de cum sunt create, compuse și

reprezentate produsele sale și când clasele de instanțiat sunt specificate la execuție

evitați construirea unei ierarhii de clase-factory paralelă cu ierarhia claselor de produse



ABSTRACT FACTORY

Scop

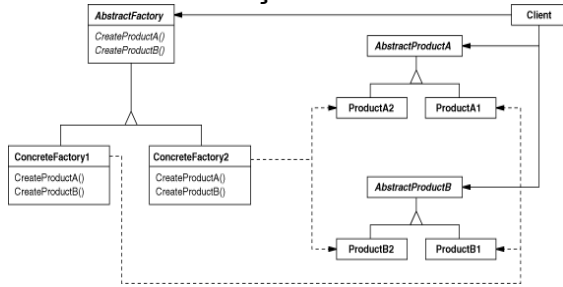
Oferă o interfață pentru crearea de familii de obiecte înrudite sau dependente fără specificarea claselor lor concrete

Aplicabilitate

Sistemul trebuie să fie independent de cum sunt create, compuse și reprezentate produsele sale

Sistemul trebuie configurat de una din mai multe familii de produse

Trebuie forțat ca o familie de obiecte produs să fie folosite împreună



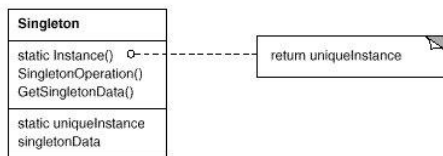
SINGLETON

Scop

Asigură ca o clasă să aibă doar o singură instanță și furnizează un punct de acces global la ea

Aplicabilitate

- dorim exact o instanță a unei clase
- accesibilitate pentru clienți dintr-un singur punct
- dorim ca instanța să fie extensibilă
- poate permite deasemenea și un set numărabil de instanțe
- optimizare față de vizibilitatea globală
- mai bine decât o clasă statică:
 - nu se poate răzgândi
 - metode niciodată virtuale



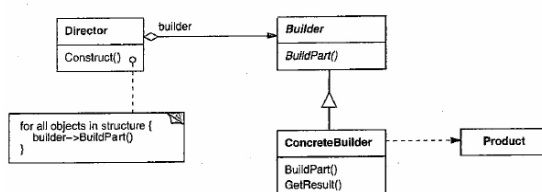
BUILDER

Scop

Separă construirea unui obiect complex de reprezentarea sa, astfel încât același proces de construcție poate genera reprezentări diferite

Aplicabilitate

- algoritmul pentru crearea obiectului complex trebuie să fie independent de părțile ce compun obiectul și de modul lor de asamblare;
- procesul de construcție trebuie să permită reprezentări diferite pentru obiectul construit;



ITERATOR

Scop

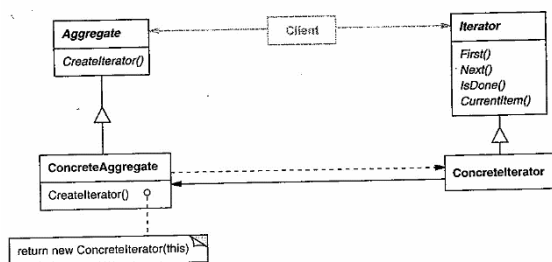
Furnizează o modalitate de accesare secvențială a elementelor unui agregat, fără a expune reprezentarea care stă la baza acestora

Aplicabilitate

Pentru a accesa conținutul unui obiect agregat fără să îi expuneți reprezentarea internă

Pentru a susține traversări multiple ale obiectelor agregat

Pentru a furniza o interfață uniformă de traversare a diferitelor structuri agregat (i.e. pentru a susține iterații polimorfice)



COMMAND

Scop

Încapsulează cereri ca obiecte, și permite:
parametrizarea clienților cu diferite cereri
înlănțuirea sau log-area cererilor
suport pentru operații anulabile (UNDO)

Aplicabilitate

Parametrizarea obiectelor

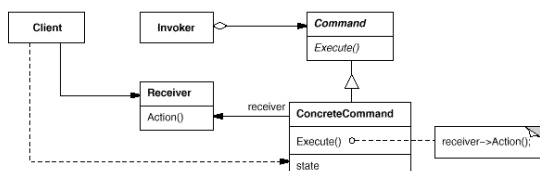
Specificarea, înlănțuirea, și executarea cererilor la diferite momente

Support undo

Modelarea tranzacțiilor

structurarea sistemelor construite pe operații primitive, în jurul operațiilor de nivel înalt

interfața comună => invocarea identică a tuturor tranzacțiilor



MEMENTO

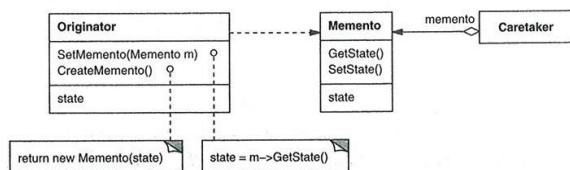
Scop

Captează și externalizează starea internă a unui obiect astfel încât obiectul poate fi restaurat în această stare mai târziu, fără a sparge încapsularea

Aplicabilitate

(o parte din) starea unui obiect trebuie salvată pentru a readuce mai târziu obiectul în această stare

O interfață directă pentru obținerea stării ar expune detalii de implementare și ar sparge încapsularea obiectului



OBSERVER

Scop

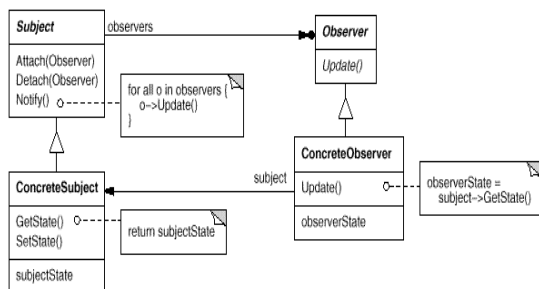
Definește o dependență de tipul unul-la-mai-mulți între obiecte astfel încât atunci când un obiect își schimbă starea, toate dependențele sale sunt anunțate și actualizate automat

Aplicabilitate

Când o abstracțiune are două aspecte, una dependentă de cealaltă; încapsularea acestor aspecte în obiecte diferite permite varierea și refolosirea lor independentă;

Când modificarea unui obiect solicită modificarea altora, și nu știm exact câte obiecte trebuie modificate (modificări dinamice, la execuție);

Când un obiect trebuie să poată notifica alte obiecte fără să presupună cine sunt acestea (CUPLAJ REDUS ÎNTRE ACESTE OBIECTE)



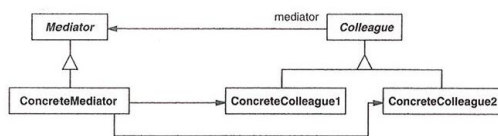
MEDIATOR

Scop

Definește un obiect ce încapsulează cum interacționează o mulțime de obiecte
Promovează cuplajul redus prin faptul că obiectele nu mai fac referință direct unele la altele, permițând varierea independentă a interacțiunii lor

Aplicabilitate

O mulțime de obiecte comunică într-un mod bine definit dar complex;
Interdependențele sunt nestructurate și dificil de înțeles;
Refolosirea unui obiect este dificilă deoarece comunică cu prea multe alte obiecte
Un comportament distribuit între mai multe clase ar trebui să fie readaptabil fără derivare



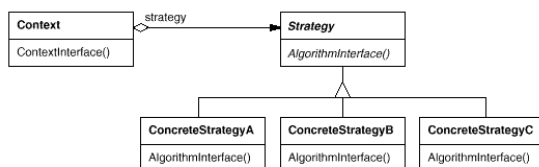
STRATEGY

Scop

Definește o familie de algoritmi, încapsulează fiecare algoritm, îi face interschimbabili
Permite algoritmului să varieze independent de clienții ce îl folosesc

Aplicabilitate

Sunt necesare variante diferite ale unui algoritm
Un algoritm folosește date despre care clienții nu trebuie să știe
evită expunerea structurilor de date complexe, specifice algoritmului
Multe clase relaționate diferă doar prin comportament
configurează o clasă cu un comportament particular



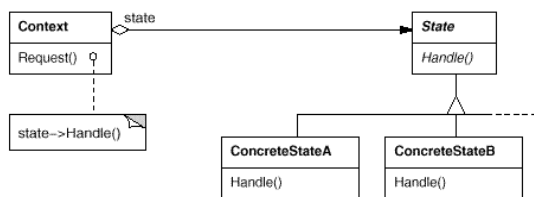
STATE

Scop

permite unui obiect să își modifice comportamentul când i se modifică starea internă obiectul va părea că își schimbă clasa

Aplicabilitate

comportamentul obiectului depinde de starea sa
trebuie să își modifice comportamentul la execuție în funcție de stare
operații cu instrucțiuni condiționale multiple, depinzând de starea obiectului
starea reprezentată de una sau mai multe constante enumerate
mai multe operații cu aceeași structură condițională



PROXY

Scop

Furnizează un surogat (substitut) pentru alt obiect pentru a controla accesul la el

Aplicabilitate

Oricând este nevoie de o referință către un obiect mai flexibilă sau mai sofisticată decât un simplu pointer

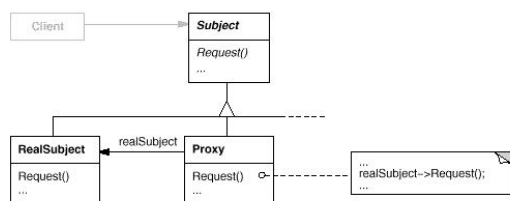
Proxy la distanță

Proxy virtual

Proxy de Protecție

Proxy -accesorii (pointeri inteligenți)

Previn ștergerea accidentală a obiectelor (numără referințele)



BRIDGE

Scop

Decuplează o abstracțiune de implementarea sa

Permite implementării să varieze independent de abstracțiunea sa

Abstracțiunea definește și implementează interfața

Toate operațiile din abstracțiune apelează metode din obiectul de implementare

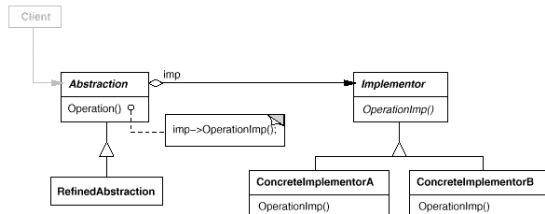
Aplicabilitate

Evită legarea permanentă între o abstracțiune și implementarea sa
Abstracțiunile și implementările lor trebuie să fie independent și extensibile prin derivare

Ascunde implementarea unei abstracțiuni complet de clienți

Codul lor nu trebuie să fie recompilat când se modifică implementarea

Partajează o implementare între mai multe obiecte și acest lucru trebuie ascuns de client



COMPOSITE

Scop

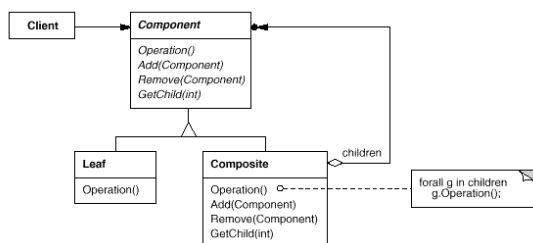
Tratează obiectele individuale și compunerile acestor obiecte uniform

Compune obiectele în structuri arborescente pentru a reprezenta agregări recursive

Aplicabilitate

reprezentarea ierarhiilor de obiecte de tip parte-întreg

abilitatea de a ignora diferența dintre compuneri de obiecte și obiecte individuale



FLYWEIGHT

Scop

Folosește partajarea pentru a suține implementarea eficientă a unui număr mare de obiecte cu granulație fină (structură complexă)- de exemplu caractere individuale sau iconițe pe ecran

Aplicabilitate

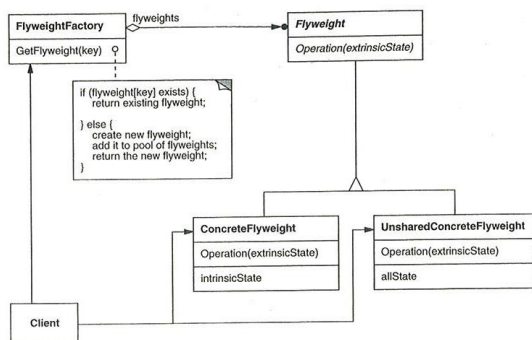
O aplicație folosește un număr mare de obiecte

Costurile de stocare sunt ridicate din cauza numărului mare de obiecte

Cea mai mare parte din starea unui obiect poate fi făcută extrinsecă

Multe grupuri de obiecte se pot înlocui de un număr relativ mic de obiecte partajate, odată ce am înlăturat starea extrinsecă

Aplicația nu depinde de identitatea obiectelor. Întrucât obiectele flyweight pot fi partajate, testele de identitate vor întoarce true pentru obiecte conceptual distincte



ADAPTER

Scop

Convertește interfața unei clase într-o altă interfață așteptată de client
 Permite unor clase cu interfețe diferite să lucreze împreună (prin “traducerea” interfețelor)

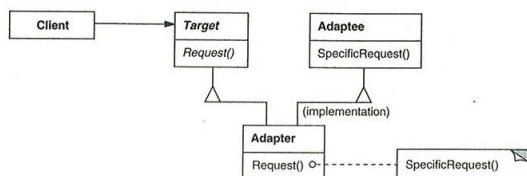
Aplicabilitate

Când dorim să folosim o clasă preexistentă, și interfața acestei nu se potrivește cu interfața de care avem nevoie

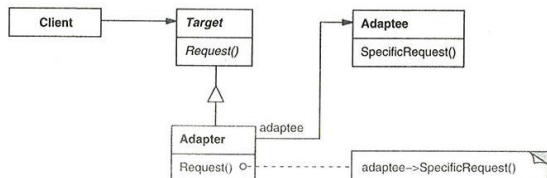
Vrem să creem o clasă reutilizabilă ce cooperează cu clase nerelaționate și neprevăzute (i.e. clase ce pot avea interfețe incompatibile)

(doar pentru adaptor obiect) Când avem nevoie să folosim mai multe subclase existente, dar nu e practic să derivăm din fiecare pentru a le adapta interfața. Un obiect adaptor poate adapta interfața clasei sale părinte.

Structura CLASA



Structura OBIECT



CHAIN OF RESPONSIBILITY

Scop

Decuplează transmitătorul cererii de primitor oferind mai multor obiecte șansa de a trata cererea

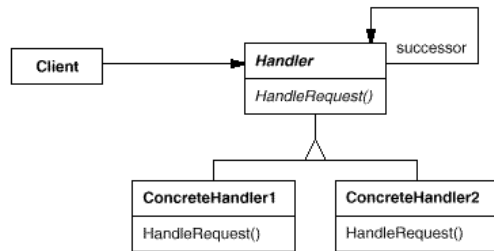
Pune primitorii într-un lanț și transmite cererea de-a lungul lanțului până ce un obiect o tratează

Aplicabilitate

mai mult decât un obiect poate trata o cerere

mulțimea de obiecte ce pot trata cererea trebuie să fie specificabilă dinamic

trimitem o cerere la mai multe obiecte fără să specificăm primitorul



DECORATOR

Scop

Adaugă responsabilități unui obiect particular, mai degrabă decât clasei sale

Atașează dinamic responsabilități adiționale unui obiect.

O alternativă flexibilă la derivare

Cunoscut și ca => Wrapper

Aplicabilitate

Adaugă responsabilități obiectelor transparent și dinamic

i.e. fără să afecteze alte obiecte

Extinderea prin derivare nu e practică

poate duce la prea multe subclase

