M. Caramihai, © 2020

PROGRAMAREA ORIENTATA OBIECT

CURS 11

Design Patterns (Sabloane software) in POO

Calitatea codului sursa

Principii urmarite in scrierea codului:

- Usor de citit/inteles – clar
- Usor de modificat –structurat
- Usor de reutilizat
- Simplu (complexitate)
- Usor de testat
- Implementeaza pattern-uri pentru problem standard

Despre sabloane

- **Definitie:** Un **sablon** reprezinta o solutie comuna a unei probleme intr-un anumit context.

Design patterns – modele, sabloane, tipare de proiectare – solutii reutilizabile la problemele de programare *soft*

Un **sablon (*design pattern*) NU** reprezinta o clasa sau o biblioteca pe care o putem integra in sistemul software

→ Este un model ce trebuie implementat in situatia corecta.

→ Au scopul de a ajuta in rezolvarea unor probleme similare cu alte probleme pentru care au fost deja identificate rezolvari.

Avantaje

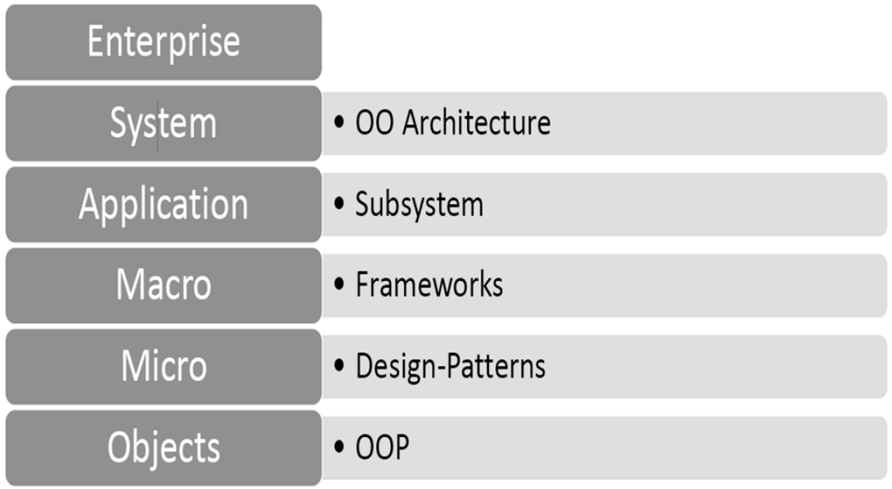
- sunt solutii folosite si testate de comunitate;
- exista module deja dezvoltate, care pot fi integrate foarte usor in proiectele de anvergura;
- reprezinta concepte universal cunoscute - definesc un vocabular comun;
- ajuta la comunicarea intre programatori;
- permit intelegerea mai facila a codului sursa/a arhitecturii;
- conduc la evitarea rescrierii codului sursa - **refactoring**.
- permit reutilizarea solutiilor standard la nivel de cod sursa/arhitectura;
- permit documentarea codului sursa/arhitecturilor

Dezavantaje

- Necunoasterea corecta a design pattern-ului conduce la ambiguitate;
- Din dorinta de aplicare a design pattern-urilor se complica foarte mult codul sursa scris;
- “Irosirea timpului” cu etapa de analiza.

Utilizare

- Identificare problema;
- **Mapare Design Pattern – Problema;**
- Identificare participanti;
- Alegerea numelor participantilor;
- Implementarea interfetelor si claselor;
- Implementarea metodelor.



Clasificare (1)

- **Dupa scop:**
  - Sabloanele **creationale** (*creational patterns*) privesc modul de creare al obiectelor.
  - Sabloanele **structurale** (*structural patterns*) se refera la compozitia claselor sau al obiectelor.
  - Sabloanele **comportamentale** (*behavioral patterns*) caracterizeaza modul in care obiectele si clasele interactioneaza si isi distribuie responsabilitătile

Clasificare (2)

- **Domeniu de aplicare:**
  - **sabloanele claselor** se refera la relatii dintre clase, relatii stabilite prin mostenire si care sunt statice (fixate la compilare).
  - **sabloanele obiectelor** se refera la relatiile dintre obiecte, relatii care au un caracter dinamic .

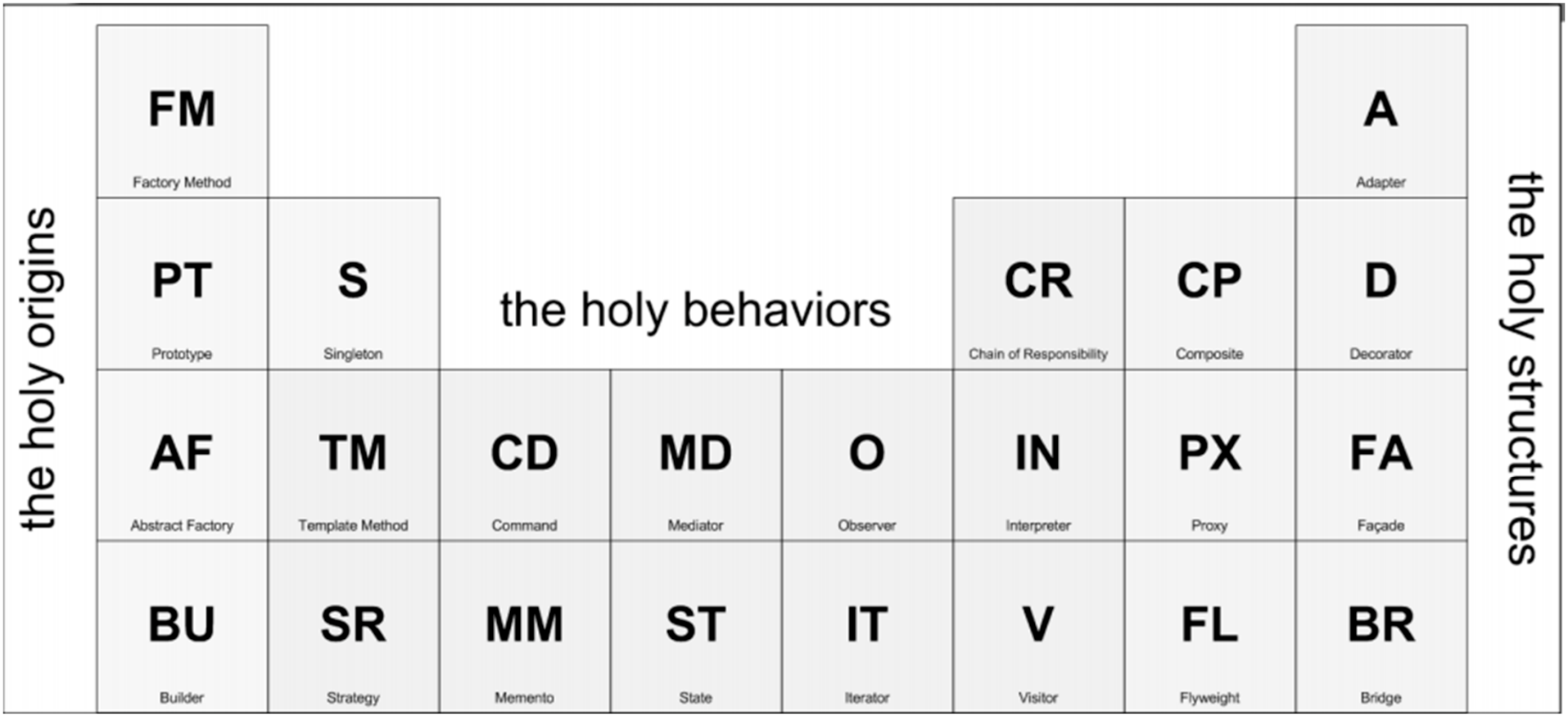
Elementele ce descriu un sablon

- **Nume**: foloseste pentru identificare; descrie sintetic problema rezolvata de sablon si solutia.
- **Problema**: descrie cand se aplica sablonul; se descrie problema si contextul.
- **Structura** – diagrama UML a claselor pentru realizarea design pattern-ului;
- **Participanti** – clasele ce fac parte din structura respectivului design pattern;
- **Implementare** – exemplu de cod sursa pentru o problema rezolvata prin acel design pattern;
- **Utilizari** – folosiri concrete si practice ale design pattern-ului;
- **Corelatii** – relatia cu alte design pattern-uri.

Cele mai importante sabloane (1)

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory	Adapter	Interpreter
				Template Method
	Object	Abstract Factory	Adapter	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Facade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

Cele mai importante sabloane (2)



Cele mai importante sabloane (3)

- Tiparele **structurale** se ocupa de obicei cu relatiile dintre entitati, facand posibil mai usor lucrul impreuna al acestor entitati.
- Tiparele **creationale** ofera mecanisme de instantiere, devenind astfel mai usoara crearea obiectelor intr-o maniera profitabila pentru situatia curenta.
- Tiparele **comportamentale** sunt folosite in comunicarea dintre entitati si face ca aceasta comunicare sa fie mai usoara si mai flexibila

Sabloane creationale

- Ajuta la initializarea si configurarea claselor si obiectelor;
- Design paternurile creationale separa crearea obiectelor de utilizarea lor concreta.

- Paternurile creationale ofera o foarte mare flexibilitate in ceea ce priveste:
  - Cine este creat;
  - Cine creaza obiectul;
  - Cum este creeat;
  - Cand este creeat.

Sabloane creationale

- **Tiparele (modelele) creationale** abstractizeaza procesul instantierii. Ele ajuta la construirea unui sistem independent de cum sunt create, compuse si reprezentate obiectele sale. O clasa creata folosind acest tipar de design foloseste mostenirea pentru a varia clasa care este instantiata pe cand un obiect creat cu acest tipar va delega instantierea unui alt obiect.

Observatii:

- Configurarea poate fi **statica** (acest lucru este specificat la momentul compilarii) sau **dinamica** (specificat la timpul rularii).

- Uneori, **modelele creationale** sunt in competitie intre ele. De ex., sunt cazuri in care fie **Prototype** fie **Abstract Factory** poate fi mai bun.

Abstract Factory

- ✓ Ofera o interfata pentru a crea familii de obiecte inrudite sau dependente fara a fi nevoiti sa specificam clasa concreta
- ✓ Ofera o ierarhie care incapsuleaza mai multe platforme posibile si posibilitatea de a construi o suita de produse
- ✓ Operatorul *new* devine daunator in instantiere (nu se mai face direct)

Factory Method

- ✓ Defineste o interfata de creare a unui obiect , dar lasa clasele derivate (subclasele) sa decida cum instantiaza obiectul.
- ✓ Ofera posibilitatea crearii de obiecte concrete dintr'o familie de obiecte, fara sa se stie exact tipul concret al obiectului.
- ✓ Se defineste un constructor "virtual"
- ✓ Operatorul *new* devine daunator, inutil

Utilizari:

- ✓ Crearea de view-uri pentru GUI.
- ✓ Existenta unei familii de obiecte intr'o aplicatie

Prototype

- ✓ Specifica tipurile de obiecte ce se pot crea utilizand o instanta "prototip" si creaza obiecte noi copiind acest prototip
- ✓ Evita costul crescut al crearii unui obiect nou in modul standard (utilizand operatorul *new*) cand aceasta este foarte costisitoare si scumpa pentru o aplicatie data.
- ✓ Pentru a implementa acest pattern, se declara o clasa abstracta de baza care specifica o metoda virtuala pura clone(). Orice clasa care este derivata din clasa de baza implementeaza metoda clone().
- ✓ Clientul, in loc sa scrie cod care invoca operatorul *new*, apeleaza metoda clone() a prototipului sau apeleaza o metoda factory cu un parametreu specificand clasa derivata dorita.
- ✓ **Utilizare**: Atunci cand obiectele create seamana intre ele, iar crearea unui obiect dureaza foarte mult sau consuma resurse foarte multe.

Singleton

- ✓ Numele semnifica faptul ca putem avea un singur obiect de tipul clasei respective: **SINGLE**.
- ✓ Clasa este cea care se ocupa de faptul ca poate fi creat un singur obiect. Nu lasa acest lucru pentru utilizatori sau apelatori.
- ✓ Mecansimul prin care este posibila impunerea unei singure clase este realizat prin definirea constructorului clasei respective ca **privat** si furnizarea unei metode (care se numeste metoda **singleton**) care este apelata in momentul in care se doreste instantierea. Aceasta va verifica numarul de instante si va controla procesul de instantiere.
- ✓ Patternurile **Abstract Factory, Builder si Prototype** pot utiliza **Singleton** in implementarile lor.
- ✓ Modelele **Singleton** sunt de obicei preferate in locul variabilelor globale deoarece:
  - ➢ nu "polueaza" spatiul de nume global cu variabile ne-necesare.
  - ➢ permit alocarea si initializarea "lenesa" (adica se amana pand cand acestea sunt intr-adevar necesare) intru-cat in cele mai multe limbaje, variabilele globale vor consuma intotdeauna multe resurse.

Singleton vs clasa statica

- **Singleton** respecta principiile de programare orientata obiect (POO);
- Un obiect **Singleton** poate fi trimis ca parametru unei functii, in schimb o clasa statica nu poate fi transmisa ca parametru;
- O clasa **Singleton** poate implementa o interfata sau extinde o alta clasa;

Singleton - caracteristici

- Deschiderea unei singure instante ale unei aplicatii.
- Conexiune unica la baza de date.

Consecinte:

- acces controlat la instanta unica
- reducerea spaţiului de nume (eliminarea variab. globale)
- permite rafinarea operatiilor si reprezentarii
- mai flexibila decât operaţiile la nivel de clasa (statice)

→ corelatii:

- **Factory** – o singura fabrica de o obiecte
- **Builder** – un singur obiect de construit alte obiecte.

Builder

- Ajuta la crearea de obiecte concrete.
- Numele vine de la faptul ca ajuta la construirea de obiecte (**build**).
- Se construiesc obiecte complexe prin specificarea anumitor proprietati dorite din multitudinea existenta.

- Implementare
  - Crearea obiectului complex in constructorul clasei Builder si modificarea atributelor conform cerintelor.
  - Crearea obiectului complex se realizeaza in metoda build() pe baza setarilor realizate.

- **Utilizare**: in general pentru construirea de obiecte complexe cu foarte multe atribute.

Sabloane structurale

- Ajuta la compunerea si configurarea claselor si obiectelor;

- Design paternurile structurale sunt concentrate pe cum sunt compuse clasele si obiectele pentru formarea de structuri complexe (obiecte complexe).

Adapter / Wrapper (1)

- Design pattern-ul **Adapter** rezolva problema utilizarii anumitor clase din framework-uri diferite, care nu au o interfata comuna.
- Clasele existente nu se vor modfica ci se va adauga noi clase pentru realizarea unui **Adapter** intre acestea.
- Clasa **Wrapper.**
- Utilizarea claselor existente se va face mascat prin intermediul adapterului creat.

- **Observatie**: Adapterulnu adauga functionalitate. Functionalitatea este realizata de clasele existente.

Adapter / Wrapper (2)

Aplicabilitate - folosim modelul Adapter atunci cand:

- ✓ Vrem sa folosim o clasa existenta si interfata pe care o folosim nu se potriveste cu cea de care avem nevoie.
- ✓ Vrem sa cream o clasa reutilizabila ce va coopera cu alte clase neprevazute
- ✓ Vrem sa folosim mai multe sub-clase existente dar este imposibil de adaptat interfata lor. (doar in cazul obiectului adaptor)

Adapter - structura

- Adapter de obiecte
  - Clasa **Adapter** contine o **instanta** a clasei existente si implementeaza interfata la care trebuie sa faca adaptarea.
  - Prin implementarea interfetei se asigura implementarea unui set de metode. Aceste metode vor face apeluri/ call-uri ale metodelor specifice clasei existente prin intermediul instantei.
- Adapter de clase
  - Clasa **Adapter** mosteneste clasa existente si implementeaza interfata la care trebuie sa faca adaptarea.
  - Prin implementarea interfetei se asigura implementarea unui set de metode. Aceste metode vor face apeluri/ call-uri ale metodelor specifice clasei existente prin intermediul parintelui (**super**).

Facade (1)

- **Usureaza** lucrul cu framework-uri foarte complexe.
- Realizeaza o **fatada** pentru aceste framework-uri, iar cine doreste sa utilizeze acele framework-uri, poate folosi aceasta fatada, fara a fi necesara cunoasterea tuturor claselor, metodelor si atributelor din cadrul framework-ului

Facade (2)

Aplicabilitate - folosim modelul Facade atunci cand:

- ✓ Vrem sa oferim o interfata simpla unui subsistem complex
- ✓ Exista multe dependente intre clienti si implementarea claselor unei abstractii
- ✓ Vrem un subsistem etajat.

Decorator (1)

- Este folosit pentru modificarea functionalitatii unui obiect la runtime.
- Este folosit de asemenea pentru adaugarea de noi functionalitati unui obiect la runtime.
- Se pot face decorari multiple prin mostenire continua.

Decorator (2)

Aplicabilitate – folosim Decorator atunci cand:

- ✓ Adaugam responsabilitati obiectelor individuale in mod dinamic si transparent
- ✓ Pentru responsabilitati ce pot fi retrase
- ✓ Cand extensii cu ajutorul sub-claselor sunt impracticabile.

Utilizari:

- Pentru adaugarea de noi functionalitati claselor existente.
- Pentru imbunatatirea claselor existente fara a modifica codul existent si fara face extindere sau mostenire

Composite (1)

- Este un design patterns structural folositatuncicandeste necesara crearea unei structuri ierahicesau o structura arborescenta prin compunerea de obiecte.

- **Observatie**: Composite nu este o structura de date (arbore).

Composite (2)

Aplicabilitate – folosim modelul Composite atunci cand:

- ✓ Vrem sa reprezentam parti intregi din ierarhia unui obiect
- ✓ Vrem clientii sa poata ignora diferentele dintre compozitii de obiecte si obiecte individuale

Utilizari:

- Meniuri ale aplicatiilor;
- Meniuri de la restaurant;
- Pentru reprezentarea oricarei arborescente.

Flyweight (1)

- Este utilizat atuncicand trebuie sa construim foarte multe obiecte/instante ale unei clase, insa majoritatea obiectelor au o parte comuna, sau permanenta.

- Astfel prin utilizarea design pattern-ului Flyweight se reduce consumul de memorie, pastrandu-se intr'o singura instanta partea comuna.

- Partea care difera de la un obiect la altul este salvata intr-o alta clasa si este adaugata dupa construirea obiectelor.

Flyweight (2)

Aplicabilitate – modelul Flyweight se foloseste atunci cand urmatoarele afirmatii sunt corecte:

- O aplicatie foloseste un numar mare de obiecte
- Costul stocarii este mai mare din cauza cantitatii mari de obiecte
- Majoritatea starilor obiectelor pot fi preparate
- Multe grupuri de obiecte pot fi inlocuite de cateva obiecte impartite odata ce starea preparata este eliminata
- Aplicatia nu depinde de identitatea obiectului.

Utilizari:

- in jocuri, atunci cand foarte multe modele seamana, insa difera prin culoare sau prin pozitie (d.e. copaci, masini, oameni, etc).

Proxy / Surrogate (1)

- Este utilizat atunci cand se doreste pastrarea functionalitatii unei clase, insa aceasta se va realiza doar in anumite conditii.

- Prin Proxy se controleaza comportamentul si accesul la un obiect.

Proxy / Surrogate (2)

Aplicabilitate – proxy este folosit atunci cand este nevoie de o referinta mai diversificata sau sofisticata a unui obiect:

- Un proxy la distanta ofera o reprezentare locala a unui obiect aflat in alt spatiu de adresa.
- Un proxy virtual creaza obiecte scumpe la comanda.
- Un proxy pentru protectie controleaza accesul la obiectul original.

Utilizare

De fiecare data cand se doreste realizarea de permisiuni pentru anumite obiecte sau pentru accesul la anumite modificari ale obiectelor.

Comparatii intre sabloanele structurale

Adapter vs Bridge

- promoveaza felxibilitatea oferind un nivel indirect unui alt obiect
- trimit mai departe cererile catre acest obiect dintr-o interfata alta decat cea proprie

- Diferente:
  - Adapter se ocupa de rezolvarea incompatibilitatii intre doua interfate existente
  - Bridge ofera o interfata stabila clientilor si chiar te lass a variezi clasele care au implementat-o

Composite vs. Decorator vs Proxy

- **Composite si Decorator :**
  - ambele se bazeaza pe compozitii recursive pentru a organiza un numar nedefinit de obiecte

- **Decorator si Proxy**
  - ambele modele descriu cum sa oferi un nivel indirect unui obiect
  - implementarea ambelor modele pastreaza o referinta unui alt obiect unde trimit cererile

Sabloane comportamentale

- **Sabloanele comportamentale (behavioral patterns)** caracterizeaza modul in care obiectele si clasele interactioneaza si isi distribuie responsabilitatile. Ele se pot aplica obiectelor sau claselor.
- De asemenea:
  - furnizeaza solutii pentru o mai buna interactiune intre obiecte si clase.
  - controleaza relatiile complexe dintre clase.
  - permit distributia responsabilitatilor pe clase si descrie interactiunea intre clase si obiecte

Strategy (1)

- Este folosit atunci cand avem mai multi algoritmi pentru rezolvarea unei probleme, iar alegerea implementarii se face la run-time.
- Fiecare comportament este dat de o clasa.
- Defineste strategia adoptata la run-time.

Strategy (2)

- Sablonul **Strategy** ajuta in alegerea unui anumit algoritm de utilizat in functie de un context specific. Sablonul contine un grup de algoritmi, fiecare din acestia fiind incapsulat intr'un obiect. Clientii ce folosesc algoritmii nu depind de acestia, variind in mod independent

Observatii:

- Sablonul Strategie defineste o familie de algoritmi, incapsuleaza fiecare algoritm si ii face interschimbabili. Acest sablon permite algoritmului sa varieze independent de clientii care il utilizeaza.
- Conform sablonului **strategy** comportamentul unei clase nu ar trebui sa fie mostenit, ci specific contextului in care ruleaza

Observer (1)

- Observer defineste o relatie de “1 : n”, in care un subiect notifica mai multi observeri.

- Acest design pattern este folosit atunci cand anumite elemente (obiecte) trebuie sa fie anuntate de schimbarea starii altor obiecte.

Observer (2)

Implementare

- in cadrul clasei concrete a subiectului observabil se gestioneaza o lista de obiecte care observa.
- in metoda de trimitere notificare se parcurge lista de observatori si se trimite un mesaj fiecaruia prin apelul functiei specifice.

Utilizare: in implementarea sistemelor distribuit

Chain of Responsibility (1)

- este un sablon comportamental ce permite evitarea cuplarii directe a expeditorului unei cereri cu un anumit destinatar, folosindu-se in acest sens clase intermediare.

- In acest fel se acorda mai multor obiecte o sansa de a rezolva cererea. Sablonul inlantuie obiectele destinatar si trece cererea de-a lungul lantului pana cand un obiect o rezolva.

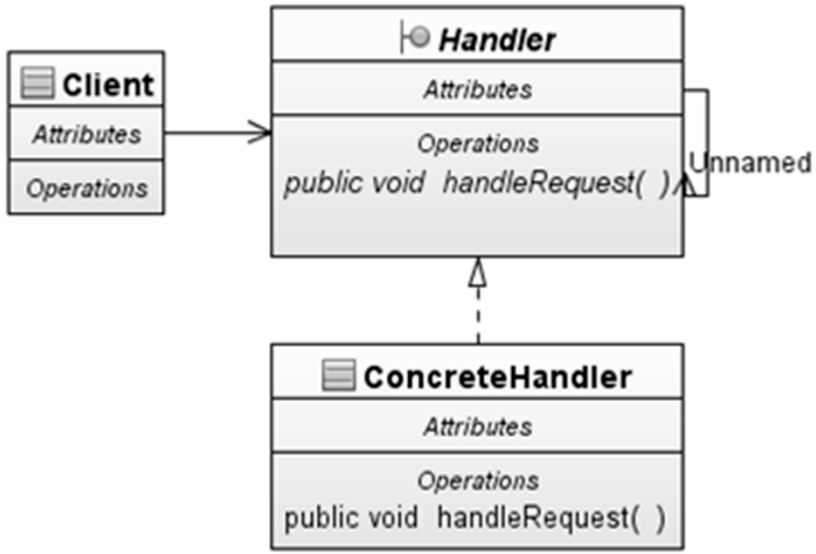
Chain of Responsibility (2)

Avantaje

- expeditorul poate sa nu cunoasca exact care este destinatarul final al cererii sale, pe el interesandu'l doar ca respectiva sarcina sa fie indeplinita;
- clasele intermediare pot alege destinatarii, gestionand eventual si gradul de solicitare al acestora si gradul de solicitare a sistemelor de calcul pe care ruleaza acestia;
- o cerere poate fi procesata de mai multi destinatari, in acelasi timp, secvential sau respectand chiar anumite fluxuri de cereri;
- clasele intermediare pot realiza log-uri ale cererilor;
- lipsa oricarui potential destinatar poate fi aflata de catre expeditor printr-un mesaj primit de la clasele intermediare.

Chain of Responsibility (3)

Exemplu:



State (1)

- State este un design pattern comportamental folosit atunci cand un obiect isi schimba comportamentul pe baza starii in care se afla.

- Este foarte asemanator cu Strategy, diferenta consta in modul de schimbare a starii respectiv a strategiei.

State (2)

- Implementare:
  - in cadrul fiecarei clase de stare exista metoda de setare a starii prin care se modifica starea contextului sau a rezervarii, in cazul de fata.
  - Modificarea starii nu se face prin setare din programul apelator ca la Strategy, ci prin apelul acestei stari.

- **Observatie**: Sablonul Stare permite unui obiect sa-si modifice comportamentul cand starea sa interna se schimba. Obiectul va parea ca isi schimba clasa.

Command (1)

- Este folosit pentru implementarea **loose coupling** (cuplari slabe).
- In acest mod ascunde aplicarea de comenzi, fara se stie concret ce presupune acea comanda. Astfel clientul este decuplat de cel ce executa actiunea.
- Conform sablonului **command** un obiect poate incapsula toate informatiile necesare pentru apelarea unei metode a altui obiect, cum ar fi: numele metodei de apelat, obiectul ce detine metoda si valorile de transmis parametrilor.

Command (2)

- **Utilizari**:
  - inregistrari de macro-uri
  - functionalitati de tip „undo”;
  - progress bar sincronizat cu executia unui grup de comenzi;
  - functionalitati de tip „wizard”;
  - lucrul cu fisiere;

- **Observatie**: Sablonul **Comanda** incapsuleaza o cerere ca obiect, permitand parametrizarea clientilor cu diferite cereri, formarea unei cozi de cereri sau stocarea istoricului acestora si asigurarea suportului pentru anularea operatiilor.

Template Method (1)

- Folosit atunci cand un algoritm este cunoscut si urmeaza anumiti pasi precisi.
- Fiecare pas este realizat de cate o metoda.
- Exista o metoda care implementeaza algoritmul si apeleaza toate celelalte metode.

- Acest sablon defineste scheletul unui algoritm dintr'o operatie, transferand unii pasi catre subclase. Sablonul permite subclaselor sa redefineasca anumiti pasi dintr-un algoritm fara a schimba structura acestuia.

Template Method (2)

Implementare

- in clasa abstracta, metoda templatese declara finala, astfel incat sa nu poata fi suprascrisa.
- in cadrul claselor concrete sunt implementate doar metodele folosite in metoda template.

Utilizari:

- Atunci cand modul de procesare sau de rezolvare a unei probleme urmeaza un numar finit si cunoscut de pasi.
- Backtracking

Memento (1)

- Sablonul amintire (**memento**) este un sablon comportamental destinat salvarii diferitelor stari curente ale unor obiecte si revenirea la aceste stari.

- **Observatie**: acest sablon captureaza si exteriorizeaza starea interna a unui obiect fara a viola incapsularea, astfel incat obiectul sa poata fi readus ulterior la respectiva stare.

Memento (2)

- Folosit atunci cand se doreste salvarea anumitor stari pentru obiectele unei clase.
- Permite salvarea si revenirea la starile salvate ori de cate ori acest lucru este dorit.
- Backup.

- **Utilizari**: Salvarea fisierelor, realizarea de backup-uri.

Visitor

- Reprezinta o operatie care va fi efectuata pe elementele unei structuri de obiecte, permitand definirea unei operatii noi fara a schimba clasele elementelor pe care opereaza.
- Utilizari:

Interpretor

- Sablonul **interpreter** descrie cum se pot interpreta expresiile intr'un anumit limbaj. El isi gaseste utilitatea in aplicatiile economice in care se doreste salvarea unor formule de calcul intr'un format accesibil utilizatorilor finali si folosirea ulterioara de catre aplicatie a acestor formule.
- Conform sablonului, atat operanzii, cat si operatorii dintr'o expresie ar trebui sa aiba aceeasi interfata.
- Un **interpretor** poate fi vazut si ca o „cutie neagra" careia i se da o expresie si care returneaza un rezultat. Se pot folosi interpretoare deja existente, cum ar fi cele pentru scripturi

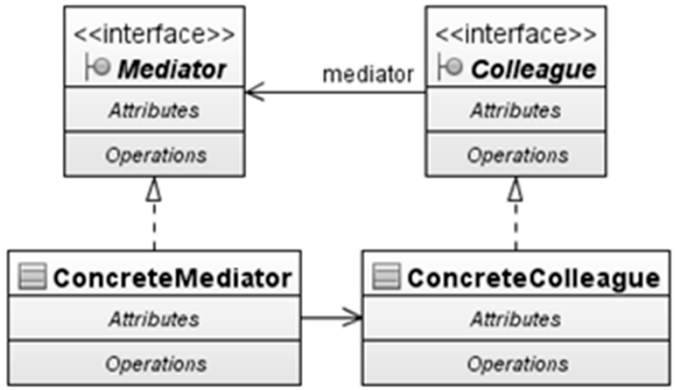
Mediator (1)

- De obicei o aplicatie este alcatuita dintr'un numar important de clase. Cu cat sunt mai multe clase intr'o aplicatie, problema comunicarii intre obiecte devine mai complexa, ceea ce face programul mai greu de citit si intretinut. Modificarea programelor, in astfel de situatii, devine mai dificila din moment ce orice modificare poate afecta codul din mai multe clase.

- Cu ajutorul sablonului **mediator** comunicatia dintre obiecte este incapsulata in obiectul mediator. Obiectele nu mai comunica direct intre ele, ci comunica in schimb prin intermediul mediatorului. Acest lucru reduce dependenta intre obiecte, micsorand cuplarea.

Mediator (2)

- Sablonul **Mediator** defineste un obiect care incapsuleaza modul in care interactioneaza un set de obiecte. Acest sablon promoveaza cuplarea slaba, interzicand obiectelor sa faca referinte explicite unul la celalalt si permite modificarea independenta a interactiunilor.



Iterator

- Sablonul **iterator** este folosit pentru a accesa elementele unui agregat (a unei colectii) in mod secvential, fara a ne folosi de caracteristi-cile acestor elemente.
- In orice moment unul din elementele colectiei este considerat ca fiind elementul curent
- **Utilizare**: in mediile de dezvoltare in care nu sunt implementate complet tipuri de date compuse (vectori, liste, stive).

Concluzii

- Sabloanele de proiectare reprezinta un set de instrumente pentru solutii varificate in rezolvarea de probleme commune în proiectarea software.
- Chiar daca nu veti intalni niciodata aceste probleme, cunoasterea sabloanelor este utila, deoarece prezinta moduri de rezolvare pentru tot felul de probleme folosind principii de design/proiectare orientat/a pe obiecte.