Motto: "Şabloanele de proiectare software te ajuta să
înveţi mai mult din succesele altora, decât din propriile eşecuri"
[Mark Johnson]

Şabloane software de proiectare

6.1. Elementele unui şablon de proiectare software

În ingineria software, prin şablon de proiectare (în engleză "design pattern") se înţelege o soluţie, un tipar care se aplică la un anumit tip de probleme.

Un design pattern este o descriere sau un model pentru o soluţie a problemei

Un şablon nu poate fi transformat direct în cod sursă.

Şabloanele de proiectare în OOD (Object Oriented Design) arată relaţiile şi interacţiunile dintre clase sau obiecte, fără a detalia clasele şi obiectele care sunt implicate.

Scopul proiectării cu şabloane este acela de a face un bun design orientat pe obiecte, şi mai mult de atât, un **design reutilizabil** (ceea ce este mai important decât reutilizarea codului, dar adesea reutilizarea designului implică şi reutilizarea codului).

Calitatea unui design software este direct proporţională cu experienţa şi cunoştinţele anterioare ale celui care îl realizează.

Un expert reutilizează soluţii pe care le-a găsit în trecut şi care deja şi-au dovedit eficienţa în probleme asemănătoare.

Când un expert găseşte o soluţie pe care o consideră bună, o refoloseşte.

Şabloanele de proiectare rezolvă probleme specifice de design, făcând proiectul OO mai flexibil, elegant şi reutilizabil. „Design patterns” ajută designerii de software propunând nişte şabloane bazate pe experienţa anterioară.

Un designer care este familiarizat cu DP, le poate aplica imediat, pentru rezolvarea unor probleme specifice, fără a fi nevoit să le redescopere.

Şabloanele software sunt structuri care respectă principiilor de proiectare, deoarece sunt obţinute ca urmare a aplicării acestor principii (şabloanele sunt rezultate directe ale principiilor).

Cu alte cuvinte, prin utilizarea şabloanelor software în cursul proiectării unei arhitecturi OO, se obţine garanţia că sistemul respectă principiile prezentate în *cursul 5* , având şi calităţile unui ”bun design”: reutilizabil, flexibil şi robust.

Lucrarea de referinţă pentru şabloanele de proiectare este ”*Design Patterns: Elements of Reusable Object-Oriented Software*” [Gamma, 1997], adesea se fac referiri la ea sub denumirea de GoF sau Gang-Of-Four.

Autorii propun soluţii recurente la probleme comune în designul software.

În cartea GoF sunt propuse 23 de şabloane de proiectare într-o formă uşor de reutilizat; aceste şabloane încorporează experienţa autorilor în rezolvarea problemelor de OOD.

Şabloanele din aceasta carte au scopul de a descrie obiectele, clasele şi modul în care acestea comunică, scopul urmărit fiind rezolvarea unei probleme generală de design într-un context particular.

Un şablon are patru elemente esenţiale:

1. **Nume**: se doreşte ca numele să descrie în câteva cuvinte, o problemă de design pattern, soluţia şi efectele ei.
2. **Problema**: descrie situaţiile în care se aplică şablonul. Explică atât problema cât şi contextul acesteia.
3. **Soluţia**: descrie elementele care fac parte din design, relaţiile dintre ele, responsabilităţile fiecărui element şi colaborările dintre acestea. Soluţia nu descrie un proiect concret specific

unei situaţii sau o implementare concretă, deoarece şablonul are un caracter general, referindu-se la o diversitate de contexte. DP furnizează o descriere abstractă a unei probleme şi un aranjament general al elementelor componente (clase şi obiecte) care rezolvă problema descrisă.

4. **Efecte**: se referă la rezultatele aplicării şablonului. Acestea sunt foarte importante atunci când evaluăm alternativele şi pentru a estima costurile şi beneficiile aplicării şablonului. Efectele unui şablon se resimt şi în flexibilitatea, extensibilitatea şi portabilitatea unui sistem.

Şablonul identifică clasele şi instanţele participante, rolul acestora, colaborarea şi distribuirea responsabilităţilor între ele.

Tabel 6.1. Şabloanele de proiectare GoF

		SCOP (Ce face şablonul)		
		Creaţional	**Structural**	**Comportamental**
Cui se aplică şablonul ?	**Clasa**	Factory Method	Adapter	Interpreter Template Method
	Obiect	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsability Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

După scop autorii GoF au propus trei mari categorii de şabloane :

Şabloanele creaţionale

Abstract Factory – furnizează o interfaţă pentru crearea familiilor de obiecte dependente sau relaţionate, fără a specifica clasele lor concrete.

Builder – separă construcţia unui obiect complex de reprezentările sale astfel încât procese de cosctrucţie similare pot crea reprezentări diferite.

Factory Method – defineşte o interfaţă pentru crearea unui obiect, dar lasă subclasele să decidă care clasă va fi instanţiată.

Prototype – specifică tipurile de obiecte create folosind o interfaţă prototipizată şi crează noi obiecte copiind acel prototip.

Singleton – asigură că o clasă are numai o instanţă, dar furnizează un punct global pentru a o accesa.

Şabloanele structurale

Adapter – face conversia interfeţei unei clase la interfaţa dorită de client. *Adapter* permite claselor să lucreze împreună, ceea ce altfel n-ar fi posibil din cauza incompatibilităţii intefeţelor.

Bridge – decupleză o formă abstractă de implementarea sa asftel încât cele două pot fi schimbate independent.

Composite – compune obiecte din arborele de structură care reprezintă parţi sau întreaga ierarhie. *Composite* lasă clienţii să trateze fiecare în parte şi uniform obiectele şi compunerea lor.

Decorator – ataşează în mod dinamic responsabilităţi adiţionale obiectelor. Furnizează subclaselor o alternativă flexibilă pentru extinderea funcţionalităţilor.

Facade – furnizează interfaţă unificată cu un set de interfeţe într-un subsistem. Defineşte o interfaţă de nivel înalt care determină ca subsistemul să fie mai uşor de utilizat.

Flyweight – foloseşte tehnica ”sharing” (divizare în mod egal) pentru a lucra cu un număr mare de obiecte mici în mod efficient.

Proxy – furnizeză un înlocuitor pentru alt obiect în scopul de a controla accesul la el.

Şabloane de comportament

Chain of Responsability- permite cuplarea expeditorului unei cereri cu destinatarul ei dând mai mult decât unui sisngur obiect şansa de a utiliza acea cerere. Înlănţuie obiecte destinatare şi lasă cererea să treacă de-a lungul şirului atât timp cât un obiect o utilizează.

Command – încapsulează cererea ca un obiect lasând prin aceasta clienţi parametrizaţi cu diferite cereri, coadă sau tabel de cereri şi suportă operaţii care se pot detaşa.

Interpreter – dându-se un limbaj, definind o reprezentare pentru gramatica sa, interpretează propoziţii din acel limbaj.

Iterator – furnizează o cale de acces la elementele unui agregat de obiecte secvenţiale fără a expune reprezentarea sa internă.

Mediator – defineşte un obiect care încapsulează toată mulţimea de obiecte cu care interacţioneză. *Mediator* promoveză cuplajul slab păstrând obiectele pentru a putea fi referite în mod explicit şi lasă să poată fi schimbată independent interacţiunea lor.

Memento – fără a fi violată încapsularea, captureză şi externează starea internă a unui obiect, astfel încât, obiectul poate fi restaurat mai târziu în starea sa.

Observer - defineşte o dependenţă „ unu-multi" între obiecte, astfel încât când un obiect îşi schimbă starea toate obiectele din dependenţa sa sunt actualizate în mod automat.

State – permite unui obiect să-şi modifice comportamentul atunci când se schimbă starea sa internă. Obiectul va părea că-şi schimbă clasa.

Strategy – defineşte o familie de algoritmi, încapsulaţi fiecare şi-i face interşanjabili. *Strategy* lasă algoritmul să poată fi schimbat în mod independent de clienţii care îl utilizează.

Template Method – lasă subclasele să-şi redefinească câţiva paşi din algoritm fară a modifica structura algoritmului.

Visitor – permite să defineşti o nouă operaţie fără schimbarea claselor de elemente cu care aceasta operează.

Acestea sunt cele mai cunoscute DP, dar nu sunt singurele.

Lucrarea amintită mai sus a fost publicată pentru prima dată în 1995, de atunci au apărut şi alte şabloane mai mult sau mai puţin cunoscute, unele noi (tratând noi tipuri de probleme, apărute ca urmare a dezvoltării limbajelor OO), altele sunt variaţii ale şabloanelor amintite mai sus (aplicaţii ale şabloanelor GoF pe cazuri particulare).

6.2. Cum rezolvă şabloanele problemele de proiectare

Şabloanele de proiectare rezolvă o serie întregă de probleme curente cu care se confruntă proiectanţii de sisteme orientate pe obiecte, în diferite moduri.

Programele orientate pe obiecte sunt constituie, evident din obiecte.

Un obiect „împachetează”, asamblează la un loc, atât datele cât şi procedurile care operează cu acele date.

Procedurile se numesc *metode* sau *operaţii*.

Un obiect execută o operaţie în momentul în care primeşte o *cerere* (sau *un mesaj*) de la un client.

Cererile constituie singura modalitate de a determina un obiect sa execute o operaţie.

Operaţiile constituie singura cale de a schimba datele interne ale obiectului.

Din cauza acestor restricţii se spune că starea internă a obiectului este *încapsulată;*ea nu poate fi accesată direct, iar reprezentarea sa este invizibilă în afara obiectului.

Partea cea mai dificilă a proiectării orientată-obiect este descompunerea proiectului în obiecte.

Sarcina este dificilă deoarece trebuie luaţi în consideraţie mulţi factori caum ar fi: încapsulare, granularitate, dependenţă, flexibilitate, performanţă, evoluţie, reutilizare, şi mulţi alţii.

Ei tot influenţează descompunerea şi câteodată sunt în contradicţie.

Multe obiecte provin din modelul de analiză, dar adesea proiectele OO au clase care nu au nici o legătură cu lumea reală.

Unele dintre acestea sunt clase de nivel inferior, cum ar fi tabelele, altele, sunt de nivel mai ridicat, cum ar fi *Compositor*, şablonul care introduce o abstractizare pentru tratarea uniformă a obiectelor care fizic nu sunt la fel.

Modelarea strictă a lumii reale conduce către un sistem care reflectă astăzi lumea reală dar nu în mod necesar şi mâine!

Abstractizările care decurg din proiectare sunt cheia realizării unui proiect flexibil.

Şabloanele de proiectare ajută la identificarea celor mai puţin evidente abstractizări şi obiecte care le capturează.

De exemplu, obiectele care reprezintă un proces sau un algoritm nu apar în natură, dar sunt parte crucială pentru proiecte flexibile.

Şablonul *Strategy* ( ) descrie cum se implementeză familii interşanjabile de algoritmi.

Şablonul *State* ( ) reprezintă fiecare stare a unei entităţi sau obiect.

Aceste obiecte sunt rareori găsite în timpul analizei sau în stadii timpurii ale proiectării.

Ele sunt descoperite mai târziu în cursul derulării proiectării când se încercă realizarea unui proiect mai flexibil sau reutilizabil.

Obiectele pot varia drastic ca număr şi dimensiune.

Se poate reprezenta orice ca obiect, plecând de la hardware şi până la întrega aplicaţie.

Cum decidem care poate fi un obiect?

Şabloanele se adresează exact acestei cerinţe.

Şablonul *Facade* descrie cum putem reprezenta subsisteme complete ca obiecte, *Flyweight* descrie cum vor fi suportate un număr uriaş de obiecte de o granularitate mai fină.

Alte şabloane descriu căi specifice de descompunere a unui obiect în altele mai mici.

6.3. Cum se selectează un şablon

Atunci când sunt mai multe şabloane la dispoziţie este relativ dificil să se selecteze exact acelea care se adresează unei probleme particulare, mai ales atunci când proiectantul nu este familiarizat cu ele.

Există câteva criterii de selecţie a celui mai potrivit şablon pentru o anumită problemă.

- Se trec în revistă şabloanele de proiectare care sunt disponibile, se studiază ce oferă fiecare şi se aleg acelea care se potrivesc cel mai bine problemei care trebuie proiectată.
- Se studiază relaţiile dintre şabloane (fig. 6.1) şi se alege grupul cel mai potrivit.
- Se studiază scopurile, asemanările şi deosebirile dintre şabloane.

- Se parcurge lista (Tabel 6.2.) cu acele aspecte ale şabloanelor care pot fi modificate în mod independent, modificări care corespund scopului proiectului propus.

Dacă proiectantul nu are experienţă în proiectarea orientată pe obiect se poate începe cu cele mai simple şi comune şabloane de proiectare: Abstract Factory, Factory Method, Adapter, Observer, Composite, Strategy, Decorator, Template Method.

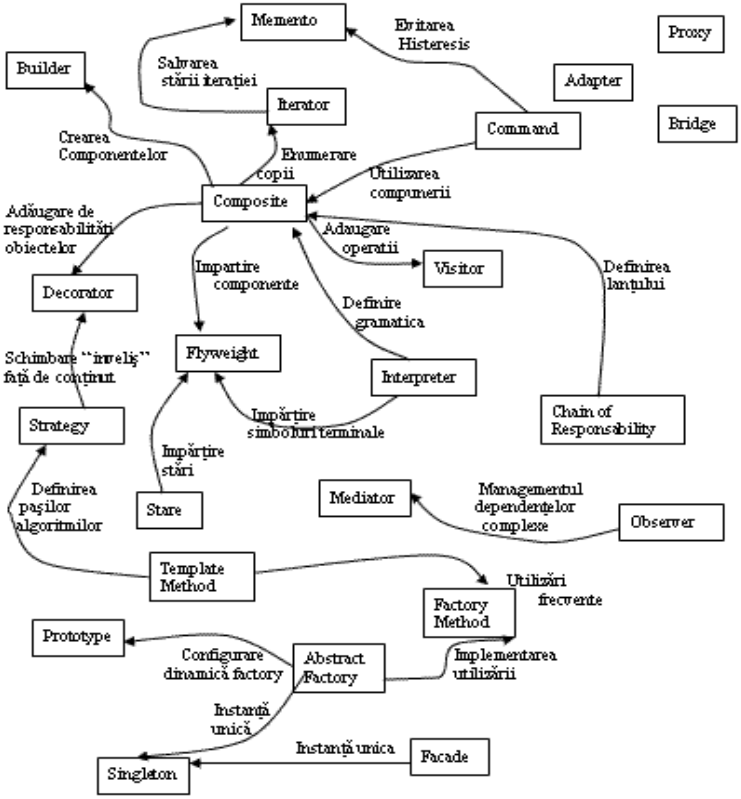

Fig. 6.1. Relaţiile între şabloanele de proiectare

6.4. Cum se foloseşte un şablon de proiectare

O abordare pas cu pas a proiectării unui sistem software folosind şabloane ar putea să decurgă în felul următor:

1. *Se citeşte descrierea şablonului pentru a avea o imagine generală*. Se acordă o atenţie deosebită secţiunilor de aplicabilitate şi consecinţe pentru a avea garanţia că şablonul de proiectare corespunde problemei;
2. *Se revine şi se studiază secţiunile de structură, participanţi şi colaborări*. Trebuie să se înţeleagă exact clasele şi obiectele din şablon şi modul în care aceastea relaţionează unele cu altele.
3. *Se examinează apoi secţiunea de cod care prezintă un exemplu concret*. Studiind codul se poate învăţa cum se implementează şablonul.
4. *Se aleg numele participanţiilor din şablon*, nume care trebuie să aibă o semnificaţie în contextul aplicaţiei. Numele participanţilor din design patterns sunt de obicei prea abstarcte pentru a apărea direct în aplicaţie şi de aceea este util să se încorporeze numele participanţilor în numele care apar în aplicaţie pentru a fi mai explicite în implementare. De

exemplu, dacă se utilizează *Strategy* pattern pentru un text care compune algoritmul, atunci putem avea clasa numită *SimpleLayoutStrategy* sau *TextLayoutStrategy.*

5. *Se definesc clasele*. Se declară interfeţele, se stabilesc relaţiile de moştenire şi se definesc variabilele care reprezintă datele şi obiectele referite. Se identifică clasele existente din aplicaţie pe care şablonul le va afecta şi se pun de acord.
6. *Se definesc numele specifice aplicaţiei pentru operaţiile din şablon*. Încă o dată, numele depind în general de aplicaţie. Se folosesc responsabilităţile şi colaborările asociate cu fiecare operaţie ca un ghid. De asemenea, este bine să existe consecvenţă în convenţiile de notare. De exemplu, se foloseşte prefixul “Creează-„ de fiecare dată când desemnăm o metodă Factory.
7. *Se implementează operaţiile având grijă de responsabilităţile şi colaborările din şablon*. Secţiunea de implemantare oferă sugestii pentru implementare.
8. *Se definesc clasele*. Se declară interfeţele, se stabilesc relaţiile de moştenire şi se definesc variabilele care reprezintă datele şi obiectele referite. Se identifică clasele existente din aplicaţie pe care şablonul le va afecta şi se pun de acord.

9. *Se definesc clasele*. Se declară interfeţele, se stabilesc relaţile de moştenire şi se definesc variabilele care reprezintă datele şi obiectele referite. Se identifică clasele existente din aplicaţie pe care şablonul le va afecta şi se pun de acord.
10. *Se definesc numele specifice aplicaţiei pentru operaţiile din şablon*. Încă o dată, numele depind în general de aplicaţie. Se folosesc responsabilităţile şi colaborările asociate cu fiecare operaţie ca un ghid. De asemenea, este bine să existe consecvenţă în convenţiile de notare. De exemplu, se foloseşte prefixul “Creează-„ de fiecare dată când desemnăm o metodă Factory.
11. *Se implementează operaţiile având grijă de responsabilităţile şi colaborările din şablon*. Secţiunea de implemantare oferă sugestii pentru implementare.

Tabelul 6.2. Aspectele de proiectare care pot fi modificate în design patterns

Scop	Şablon de proiectare	Aspecte care pot fi modificate
Creaţional	Abstract Factory	Familiile de obiecte produs
	Builder	Modalitatea în care pot fi create obiectele compozite
	Factory Method	Subclasa obiectelor care sunt instanţiate
	Prototype	Clasa obiectelor care este instanţiată
	Singleton	O singură instanţă a unei clase
Structural	Adapter	Interfaţa cu un obiect
	Bridge	Implementarea unui obiect
	Composite	Structura şi compoziţia unui obiect
	Decorator	Responsabilităţile proprii ale unui obiect fără subclase
	Facade	Interfaţa cu un subsistem
	Flyweight	Memorarea costurilor obiectelor
	Proxy	Modul în care este accesat obiectul- locaţia sa
Comportamental	Chain of Responsability	Obiectul care poate îndeplini o solicitare
	Command	Când şi cum poate fi indeplinită o solicitare
	Interpreter	Gramatica şi interpretarea unui limbaj
	Iterator	Cum sunt accesate şi parcurse elementele unei mulţimi
	Mediator	Cum şi care obiecte interacţionează unele cu altele
	Memento	Ce informaţie privată este memorată în afara obiectului şi când.
	Observer	Numărul obiectelor care depind de alt obiect; Cum poate fi

		actualizat obiectul dependent.
	State	Stările unui obiect
	Strategy	Un algoritm
	Template Method	Paşii unui algoritm
	Visitor	Operaţiile care pot fi aplicate obiectului (obiectelor) fără schimbarea clasei (claselor).