ŞABLOANE DE PROIECTARE

10 octombrie 2023

Cuprins

Capitolul 1

Prezentare generală

Scopul şabloanelor de design este de a înregistra experienţă referitoare la programarea obiect-orientată. Fiecare astfel de şablon numeşte, explică şi evaluează un design în sisteme obiect orientate.

1.1 Ce este un şablon de proiectare?

Prima oară, în literatura de specialitate conceptul de şablon (pattern) a apărut în anul 1977 când Christopher Alexander a publicat cartea "A Pattern Language". În cartea sa el nu a scris despre şabloane de proiectare în domeniul informaticii, ci despre moduri de a automatiza procesul de realizare a unei arhitecturi de succes.

El dă şi prima definiţie a ceea ce ar putea însemna un şablon de proiectare: *"Fiecare şablon descrie o problemă care tot apare în mediul nostru, după care descrie esenţa soluţiei acelei probleme, în aşa fel încât ea să poată fi folosită de mii de ori, fără a mai face acelaşi lucru de mai multe ori."*

Această carte a lui care a promovat ideea de şablon în arhitectură i-a făcut pe cei care se ocupau de industria software să se întrebe dacă acelaşi lucru nu ar putea fi valabil şi pentru domeniul informaticii.

Întrebarea care se punea era următoarea: dacă se poate afirma că un design este bun, cum se poate face ca prin anumiţi paşi să putem automatiza obţinerea unui astfel de design?

Tot Christopher Alexander este cel care a adus în prim plan ideea că, în general, în viaţa de zi cu zi, aceste şabloane se compun unele cu altele

conducând la soluţii mai complexe care agregă mai multe şabloane.

Momentul care a condus la popularizarea pe scară largă a noţiunii de şablon de proiectare în domeniul informaticii a fost apariţia în 1995 a cărţii *"Design patterns - Elements of Reusable Object Oriented Software"*, carte scrisă de E. Gamma, R. Helm, Johnsson şi Vlissides.

În această carte cei patru autori nu au meritul de a fi inventat şabloanele de proiectare pe care le-au prezentat ci mai degrabă ei au documentat ceea ce deja exista în sistemele soft ale vremii respective. Principalele lor merite sunt următoarele:

- au introdus ideea de şablon în industria software
- au catalogat şi descris fiecare şablon în parte
- au prezentat într-un mod logic, strategiile care stau la baza acestor şabloane de proiectare.

Această carte este una din cele mai de succes cărţi de informatică din toate timpurile. Chiar şi la aproximativ 15 ani de la apariţia ei, ea este citată şi utilizată la fel de mult ca la început, dacă nu chiar şi mai mult.

În semn de apreciere, autorii acesteia sunt cunoscuţi sub numele de "Gang of Four" (GoF).

Nu numai cei patru care au scris această carte au un merit deosebit la popularizarea şabloanelor de proiectare. Există şi alte persoane, care sunt poate chiar mai importante datorită faptului că ele au introdus aceste concepte. Nume precum Kent Beck, Ward Cunningham şi James Coplien au contat foarte mult în dezvoltarea acestui domeniu.

O altă definiţie a unui şablon de design, care este dată chiar de către cei patru sună astfel: *"Şabloanele de proiectare sunt descrieri ale unor obiecte şi clase care comunică şi care sunt particularizate pentru a rezolva o problemă generală de design într-un context particular"*.

În continuare prezentăm o ultimă definiţie care îi aparţine tot lui Christopher Alexander. Chiar dacă ea a fost gândită referitor la arhitectură ea este foarte potrivită în domeniul informaticii: *"Fiecare şablon de proiectare este o regulă în trei părţi, care exprimă o relaţie între un context, o problemă şi o soluţie"*.

După cum vom vedea în continuare, pentru a descrie un şablon de proiectare, trebuie neapărat să specificăm următoarele lucruri:

- *numele şablonului* - este important deoarece el ne ajută să comunicăm mai uşor cu cei cu care lucrăm
- *problema* - descrierea situaţiei în care se aplică
- *soluţia* - modalitatea de a rezolva acea problemă
- *consecinţele* - avantajele şi dezavantajele acelei abordări

1.2 De ce sunt utile?

Până acum am văzut ce sunt şabloanele de proiectare. O altă întrebare legitimă în acest moment ar fi, de ce sunt ele utile. Oare este neapărat necesar să înţelegem conceptul de şablon de proiectare? Este neapărat necesar să cunoaştem exemple de şabloane de proiectare?

Răspunsul autorului este că da, din mai multe motive:

- Utilizarea şabloanelor de proiectare conduce la reutilizarea unor soluţii care şi-au arătat de-a lungul timpului eficienţa. După cum se ştie una din cele mai importante probleme în domeniul informaticii ţine de reutilizabilitate. Dacă până acum am tot vorbit despre reutilizarea codului, oare nu se poate ca reutilizarea ideilor să fie la fel de importantă?
- Permit stabilirea unei terminologii comune. Practic, printr-un singur cuvânt se poate ca să comunicăm ceea ce altfel ar fi destul de greu de prezentat.
- Oferă o perspectivă mai înaltă asupra analizei şi designului sistemelor obiect orientate.
- Au ca principal obiectiv crearea de cod flexibil, uşor de modificat. Scopul lor este ca, în măsura în care este posibil, să permită adăugarea de noi funcţionalităţi fără a modifica cod existent.
- Odată înţelese bine, ele sunt nişte exemple foarte bune relativ la principiile de bază ale programării obiect orientate.
- Permit însuşirea unor strategii îmbunătăţite care ne pot ajuta şi atunci când nu lucrăm cu şabloanele de proiectare:

- Lucrul pe interfeţe nu pe implementări
- Favorizarea compoziţiei în dauna moştenirii
- Găsirea elementelor care variază şi încapsularea lor

1.3 Clasificarea şabloanelor

În funcţie de nivelul la care apar, şabloanele sunt de mai multe feluri:

- *idioms* - sunt primele care au apărut şi sunt dependente de anumite tehnologii (de exemplu, lucrul cu smart pointers în limbajul C++).
- *design patterns* - reprezintă soluţii independente de un anumit limbaj, putem spune că sunt un fel de microarhitecturi (ele sunt cele care vor fi prezentate în continuare).
- *framework patterns* - sunt şabloane la nivel de sistem, adică sunt şabloane care sunt folosite pentru a descrie la nivel înalt arhitectura unui întreg sistem.

Există şabloane care de-a lungul timpului au evoluat, de la prima categorie către ultima: MVC (Model View Controller).

Din punctul de vedere al scopului lor, şabloanele de proiectare se împart în 3 categorii:

- creaţionale - abstractizeaza procesul de creare a obiectelor pentru a creşte flexibilitatea designului
- structurale - permit gruparea obiectelor în structuri complexe
- comportamentale - permit definirea unui cadru optim pentru realizarea comunicării între obiecte.

În cele ce urmează vom prezenta, pe scurt, un idiom, pentru a putea face ulterior o comparaţie între el şi şabloanele de proiectare.

Partea I

Şabloane creaţionale

Capitolul 2

Singleton

Scop

Asigură faptul că o clasă are o singură instanţă şi oferă un singur punct de acces global la ea.

Motivare

În anumite situaţii vrem ca o clasă să aibă o singură instanţă care să poată fi accesată de oriunde dintr-o aplicaţie.

Am putea folosi în acest caz o variabilă globală, dar atunci am putea crea oricâte instanţe ale acelei clase.

Am putea de exemplu să avem în obiect un câmp static care să numere câte obiecte au fost create şi să genereze eroare când vrem să creem unul nou, dar acest lucru nu este atât de elegant, precum soluţia pe care o vom prezenta în continuare şi nici nu rezolvă problema accesului global la acea instanţă.

Aplicabilitate

Aplicăm acest şablon de proiectare atunci când:

- Vrem să existe o singură instanţă a unei clase, care să poată fi accesibilă clienţilor printr-un punct de acces binecunoscut.

- Când singura instanţă trebuie să poată fi extensibilă prin subclasare şi clienţii ar trebui să fie capabili să folosească o instanţă extinsă fără a-şi modifica codul.

Structură

Structura şablonului de proiectare Singleton este prezentată în figura 2.1.

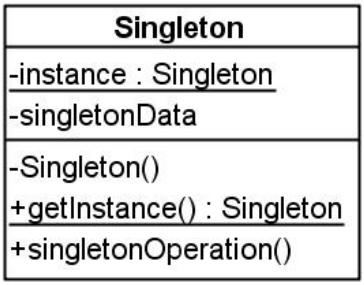

Figura 2.1: Structura şablonului Singleton

După cum se poate observa, pentru a rezolva această problemă trebuie să facem întâi şi întâi astfel încât să nu putem crea obiecte de tipul Singleton. Acest lucru se face prin realizarea unui constructor privat.

Un obiect de tip Singleton este reţinut într-o instanţă statică de tip Singleton. Foarte important este rolul metodei statice `getInstance()` care arată astfel:

```
public static Singleton getInstance()
{
  if(instance==null)
    instance=new Singleton();
  return instance;
}
```

Astfel, dacă atunci când este cerut obiectul acesta există, atunci el este doar returnat fără a se creea altul. Dacă el nu există, atunci este creat şi apoi returnat.

Singura modalitate prin care se poate accesa obiectul unic de tip Singleton este prin intermediul metodei `getInstance()`.

Folosirea acestui şablon are mai multe consecinţe:

- Prin faptul că nu se foloseşte o variabilă globală, se asigură nepoluarea spaţiului de nume cu o nouă denumire.
- Permite subclasarea unei clase de tip Singleton, astfel încât noua clasă să poată avea ea însăşi o singură existenţă.
- Cu anumite modificări se poate obţine o nouă clasă care poate crea un număr limitat de obiecte, dar mai mult decât unul.

Exemplu

În cele ce urmează este prezentat un exemplu care ilustrează modul în care se poate folosi acest şablon de proiectare. Am construit în continuare o clasă `Printer` care simulează o imprimantă. Pentru această clasă trebuie să avem o singură instanţă la un moment dat, lucru care este asigurat de folosirea şablonului de proiectare *Singleton*.

```
public class Printer {
  private static Printer instance;

  private Printer()
  {
  }

  public synchronized static Printer getPrinter()
  {
    if(instance==null)
      instance=new Printer();
    return instance;
  }

  public synchronized void print(String str)
  {
    System.out.println("Text de tiparit: "+str);
    try {
      Thread.sleep(2000);
```

```
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println("Terminat de tiparit...");
  }
}

class TiparireThread extends Thread
{
  private String textDeTiparit;
  public TiparireThread(String str)
  {
    textDeTiparit=str;
  }

  public void run()
  {
    Printer p=Printer.getPrinter();
    p.print(textDeTiparit);
  }
}

public class TestSingleton {
  public static void main(String[] args) {
    TiparireThread t1=new TiparireThread("primul text...");
    TiparireThread t2=new TiparireThread("al doilea text...");
    t1.start();
    t2.start();
  }
}
```

Partea II

Şabloane structurale

Partea III

Şabloane comportamentale