Șabloane de Proiectare

Design Patterns

Șabloane de Proiectare

- Arhitectul C. Alexander. A Pattern Language, 1977
- Gamma, Helm, Johnson, and Vlissides – *Design Patterns, Elements of Reusable Object-Oriented Software* (cunoscuta si sub numele “Gang of Four”), 1994

Ce este un sablon de proiectare?

- Definiția lui Alexander: "Fiecare şablon descrie o problemă care apare mereu în domeniul nostru de activitate şi indică esenţa soluţiei acelei probleme într-un mod care permite utilizarea soluţiei de nenumărate ori în contexte diferite"
- Un **sablon** reprezintă o soluţie comună a unei probleme într-un anumit context.

La ce sunt bune şabloanele?

- Ne usurează viața….
- Ne ajută să organizăm mai bine treaba
- Putem sa înțelegem “mai bine” POO
- Uneori pot reduce ordinul de complexitate al problemei...
- … sau fac definițiile obiectelor mai ușor de înțeles…

Clasificarea şabloanelor (I)

- **După scop:**
  - Şabloanele **creaţionale** (creational patterns) privesc modul de creare al obiectelor.
  - Şabloanele **structurale** (structural patterns) se referă la compoziţia claselor sau al obiectelor.
  - Şabloanele **comportamentale** (behavioral patterns) caracterizează modul în care obiectele şi clasele interacţionează şi îşi distribuie responsabilităţile

Clasificarea şabloanelor (II)

- **Domeniu de aplicare:**
  - **Şabloanele claselor** se referă la relaţii dintre clase, relaţii stabilite prin moştenire şi care sunt statice (fixate la compilare).
  - **Şabloanele obiectelor** se referă la relaţiile dintre obiecte, relaţii care au un caracter dinamic .

Cele mai importante şabloane

Scop / Dom. de aplicare	Creationale	Structurale	Comportamentale
Clasa	Factory Method	Adapter (clasa) Interface Marker Interface	Interpreter Template Method
Obiect	Immutable Abstract Factory Builder Prototype Singleton	Delegation Adapter (obiect) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Elementele ce descriu un şablon

- **Nume**: foloseşte pentru identificare; descrie sintetic problema rezolvată de şablon şi soluţia.
- **Problema**: descrie când se aplică şablonul; se descrie problema şi contextul.
- **Solutia**: descrie elementele care intră în rezolvare, relaţiile între ele, responsabilităţile lor şi colaborările între ele.
- **Consecinţe si compromisuri**: implicaţiile folosirii şablonului, costuri şi beneficii. Acestea pot privi impactul asupra flexibilităţii, extensibilităţii sau portabilităţii sistemului, după cum pot să se refere la aspecte ale implementării sau limbajului de programare utilizat. Compromisurile sunt de cele mai multe ori legate de spaţiu şi timp.

Structura unui șablon

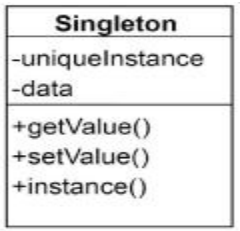
- nume si clasificare
- intentie
- cunoscut de asemenea ca
- motivatie
- aplicabilitate
- structura
- participanti
- colaborari
- consecinte
- implementare
- cod
- utilizari cunoscute
- sabloane cu care are legatura

Șablonul Singleton (I)
(Clase cu o singură instanță)

- Intenţia
  - proiectarea unei clase cu un singur obiect (o singura instanţă)
- Motivaţie
  - într-un sistem de operare:
    - exista un sistem de fişiere
    - exista un singur manager de ferestre
- Aplicabilitate
  - când trebuie să existe exact o instanta
  - clientii clasei trebuie sa aiba acces la instanta din orice punct bine definit

Șablonul Singleton (II)
(Clase cu o singură instanță)

- Structura:



- participant: Singleton
- colaborari: clientii clasei

Șablonul Singleton (III)
(Clase cu o singură instanță)

- Consecinţe
  - acces controlat la instanta unica
  - reducerea spaţiului de nume (eliminarea variab. globale)
  - permite rafinarea operaţiilor si reprezentării
  - permite un numar variabil de insţante
  - mai flexibila decât operaţiile la nivel de clasă (statice)
- Implementare

Șablonul Singleton (IV)
(Clase cu o singură instanță)

```
1.  #include <iostream.h>
2.  using namespace std;
3.  class Singleton {
4.    public:
5.      static Singleton & instance() {
6.        return uniqueInstance;
7.      }
8.      int getValue() {
9.        return data;
10.     }
11.     void setValue(int value) {
12.       data = value;
13.     }
14.   private:
15.     static Singleton uniqueInstance;
16.     int data;
17.     Singleton(int d):data(d) {
18.     }
19.     Singleton & operator=(Singleton & ob);
20.     Singleton(const Singleton & ob);
21. };
```


Șablonul Singleton (IV)
(Clase cu o singură instanță)

```
22. Singleton Singleton::uniqueInstance(100);
23. int main(){
24.     Singleton & s1 = Singleton::instance();
25.     cout << s1.getValue() << endl;
26.     Singleton & s2 = Singleton::instance();
27.     s2.setValue(9);
28.     cout << s1.getValue() << endl;
29.     return 0;
30. }
```

- Output:

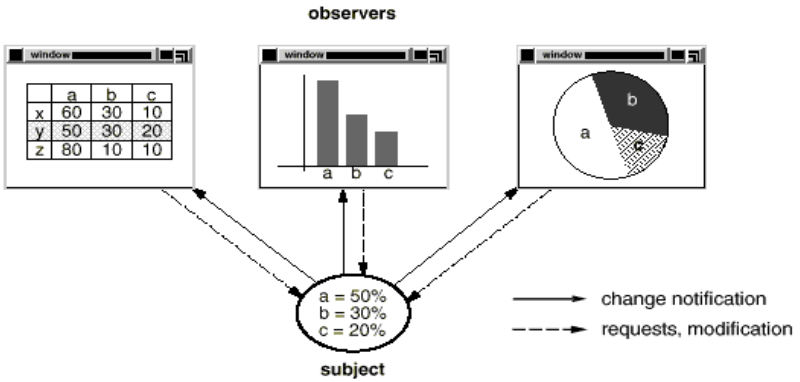
```
100
9
```


Şablonul Observator (I)

- Intenţia
  - Defineşte o relaţie de dependenţă 1..* între obiecte astfel încât când un obiect îşi schimbă starea, toţi dependenţii lui sunt notificaţi şi actualizaţi automat

Şablonul Observator (II)

- Motivaţia

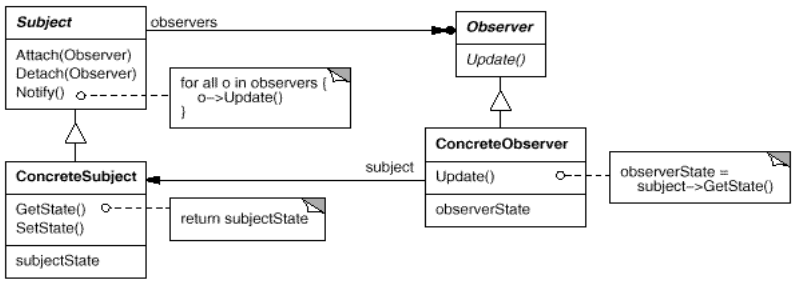


Şablonul Observator (III)

- Aplicabilitate
  - când o abstracţie are doua aspecte, unul depinzând de celalalt. Incapsulând aceste aspecte in obiecte separate, permitem reutilizarea lor in mod independent
  - când obiect necesita schimbarea altor obiecte si nu ştie cat de multe trebuie schimbate
  - când un obiect ar trebui sa notifice pe altele, fara sa ştie cine sunt acestea
  - in alte cuvinte, nu dorim ca aceste obiecte sa fie cuplate strâns (a se compara cu relaţia de asociere)

Şablonul Observator (IV)

- Structura

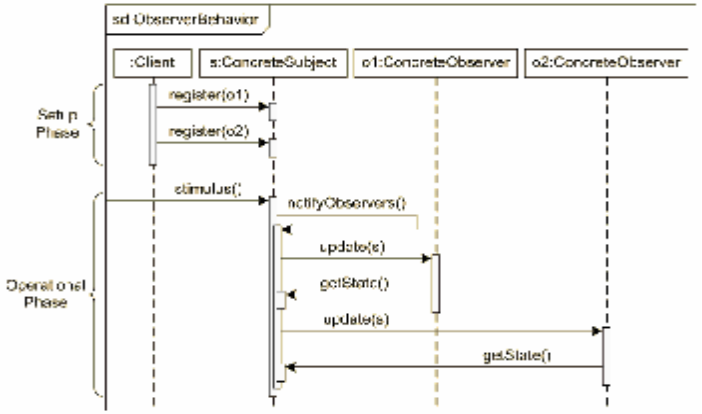


Şablonul Observator (V)

- **Participanti**
  - Subject
    - cunoaste observatorii(numarar bitrar)
  - Observer
    - defineste o interfata de actualizare a obiectelor ce trebuie notificate de schimbarea subiectelor
  - ConcreteSubject
    - memoreaza starea de interes pentru observatori
    - trimite notificari observatorilor privind o schimbare
  - ConcreteObserver
    - mentine o referinta la un obiect ConcreteSubject
    - memoreaza starea care artrebui sa fie consistenta cu subiectii

Şablonul Observator (VI)

- **Colaboratori**



Şablonul Observator (VII)

- **Consecinte**
  - abstractizeaza cuplarea dintre subiect si observator
  - suporta o comunicare de tip “broadcast”
    - notificarea ca un subiect si-a schimbat starea nu necesita cunoasterea destinatarului
  - schimbari “neasteptate”
    - o schimbare la prima vedere inocenta poate provoca schimbarea in cascada a starilor obiectelor

Şablonul Observator (VIII)

- **Implementare**
  - maparea subiectilor la observatori
    - memorarea de referinte la observatori
  - observarea mai multor subiecti
  - cine declanseaza o actualizare
    - subiectul apeleaza o metoda Notify() dupa fiecare schimbare
    - clientii sunt responsabili de apela Notify()
    - fiecare solutie are avantaje si dezavantaje (care?)
  - evitarea de referinte la subiecti stersi
    - subiectii ar trebui sa notifice despre stergerea lor (?)
    - ce se intampla cu un observator la primirea vestii?

Şablonul Observator - Cod

```
1.  #include <iostream>
2.  #include <vector>
3.  #include <string>
4.
5.  class Subject;
6.
7.  class Observer {
8.    public:
9.      virtual void Update(Subject * s) = 0;
10.      virtual ~ Observer() {
11.     };
12. };
13.
```


Şablonul Observator - Cod

```
14. class Subject {
15.     std::vector < Observer * >observers;
16.   protected:
17.     virtual void Notify() {
18.         std::vector < Observer * >::iterator iter;
19.         for (iter = observers.begin(); iter != observers.end(); ++iter)
20.             (*iter)->Update(this);
21.   } public:
22.      virtual ~ Subject() {
23.     };
24.     void Attach(Observer * o) {
25.         observers.push_back(o);
26.     }
27.     void Detach(Observer * o) {
28.         std::vector < Observer * >::iterator iter;
29.         for (iter = observers.begin(); iter != observers.end(); ++iter)
30.             if (*iter == o) {
31.                 observers.erase(iter);
32.                 return;
33.             }
34.     }
35. };
```

Şablonul Observator - Cod

```
37.  class CursValutar:public Subject {
38.    protected:
39.      float valoare;
40.    public:
41.     CursValutar(float v = 0):valoare(v) {
42.      } void setCurs(float valoare) {
43.          this->valoare = valoare;
44.          Notify();
45.      }
46.      float getCurs() {
47.          return valoare;
48.      }
49.  };
```


Şablonul Observator - Cod

```
51.  class Banca:public Observer {
52.      std::string nume;
53.    public:
54.      Banca(std::string _nume):nume(_nume) {
55.      } void Update(Subject * s);
56.  };
57.
58.  void Banca::Update(Subject * s)
59.  {
60.      CursValutar *c = (CursValutar *) s;
61.      std::cout << "Banca " << nume << " a fost notificata ca noua valoare a
   cursului este: " << c->getCurs() << std::endl;
62.  }
63.
```


Şablonul Observator - Cod

```
64.  int main() {
65.      CursValutar c(4.2);
66.      Banca b1("BRD");
67.      Banca b2("BCR");
68.      c.Attach(&b1);
69.      c.Attach(&b2);
70.      c.setCurs(10);
71.      c.Detach(&b1);
72.      c.setCurs(4);
73.      return 0;
74.  }
```

nOutput

```
Banca BRD a fost notificata ca noua valoare a cursului este: 10

Banca BCR a fost notificata ca noua valoare a cursului este: 10

Banca BCR a fost notificata ca noua valoare a cursului este: 4
```


Bibliografie