Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatica și Micorelectronică

Catedra Automatica și Tehnologii Informaționale

**Raport**

Lucrarea de laborator nr. 3

# La disciplina: Ingineria Produselor Program

Tema: Șabloane comportamentale

A efectuat: st.gr.TI-143 Cornita Constantin

A verifica: lector asistent, Chetrusca Ecaterina

Chișinău 2017

**Scopul**

De studiat și de implimentat 5 șabloane de comportament.

**Considerații teoretice**

Șabloanele de comportament tratează întrebări referitoare la conexiunile dintre obiectele și împărțirea responsabilităților între ele. Pentru aceasta, se utlizează mecanisme bazate atât pe moștenire cît și pe compoziției.

**Command Pattern**

Șablonul comandă incapsulează o cerere ca obiect, permițînd:

* aplicația definește acțiuni parametrizabile ce pot fi executate mai târziu fără a solicita clientului cunoașterea detaliile interne necesare execuției;
* pentru a nu bloca clientul, se dorește ca aceste acțiuni să fie definite și trimise spre execuție fără a mai fi gestionate de client;
* se decuplează execuția întârziată (ulterioara) a unei acțiuni de proprietar. Din punctul acestuia de vedere, acțiunea a fost deja trimisa spre execuție;
* concept echivalent cu macro-urile. Obiectul de tip command încapsulează toate informațiile necesare execuției acțiunii mai târziu de către responsabil;
* clientul este decuplat de cel ce executa acțiunea.

În figura 1 este reprezentă diagrama de clasă pentru șablonul Command.

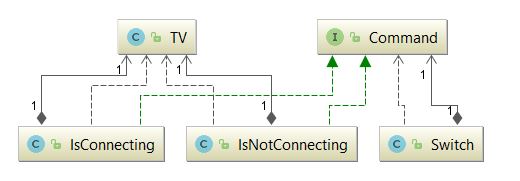


Fig.1 – Command Pattern

**Iterator Pattern**

Iterator - este un șablon comportamental, care dă posibilitatea de să treacă prin toate elementele unui obiect compus. Una dintre cele mai importante condiții pentru punerea în aplicare a acestui șablon este că un iterator trebuie să asigure integritatea obiect intern.

În figura 2 este reprezenta diagrama UML de clase pentru Iterator Pattern, precum și relațiile dintre clasele care formează acest șablon.

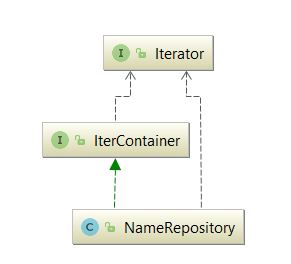


Fig.2 – Iterator Pattern

**Mediator Pattern**

Acest sablon lucrează cu interactiunea intre mai multe obiecte, permitand ca obiectele respective sa nu se refere unul pe altul in mod explicit. In felul acesta, la nevoie se poate modifica mai usor modul in care interactioneaza obiectele.

"Mediator" definește interfața pentru schimbul de informații cu obiectele "colegi", el coordonează acțiunea dintre obiectele "colegi".

Fiecare clasă "Colegi" știe despre subiectul său "Mediatorul," respectiv toți "Colegii" schimbă informația numai cu mediatorul, dar în absența acestuia, ele ar trebui să facă schimb de informații în mod direct.

"Colegii" pot trimite solicitări și primi cereri de mediator de la el. "Mediator" implementează comportamentul de cooperare prin trimiterea unei cereri către unul sau mai mulți "colegi".

Șablonul Mediator se aplică în situațiile în care:

* mulțime de obiecte comunică într-un mod bine definit, dar complex, rezultînd o interdependență nestructurată și dificil de ințeles;
* reutilizarea unui obiect este dificilă din cauza că el se referă și comunică cu multe alte obiecte;
* un anumit comportament distribuit pe mai multe clase trebuie să poată fi adaptat fără a implica o mulțime de subclase.

În figura 3 este reprezentată diagrma de clase pentru Șablonul Mediator, de asemenea sutn reprezentate si clasele participante la crearea acestui șablon.

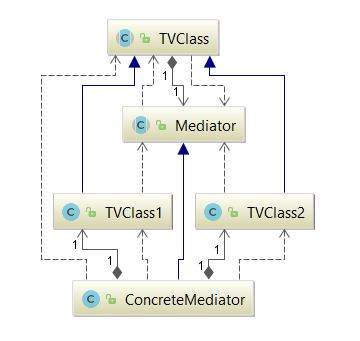


Fig.3 – Mediator Pattern

**Strategy Pattern**

Şablonul Strategie defineşte o familie de algoritmi, încapsulează fiecare algoritm şi îi face interschimbabili.

Acest şablon permite algoritmului să varieze independent de clienţii care îl utilizează.

Acest șablonm se utilizeaza atunci cînd:

* mai multe clase înrudite diferă doar prin modul lor de comportare;
* este nevoie de variante diferite ale unui algoritm;
* un algoritm utilizează date ce nu trebuie cunoscute de către clienţi;
* Când o clasă defineşte mai multe comportamente iar acestea apar în operaţiile ei sub formă de instrucţiuni condiţionale multiple.

In figura 4 este reprezentată diagrama de clase al șablonului Strategie. La fel sunt repărezentatate și clasele prin intermediul cărora este alcatuit acest șablon.

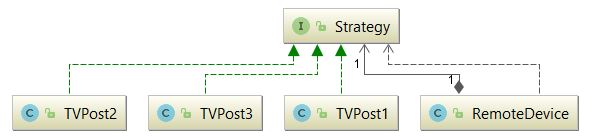


Fig. 4 – Strategy Pattern

**Visitor Pattern**

Şablonul Vizitator reprezintă o operaţie care va fi efectuată pe elementele unei structuri de obiecte, permiţând definirea unei operaţii noi fără a schimba clasele elementelor pe care operează.

Acest șablon, se aplică atunci cînd:

* când o structură de obiecte conţine multe clase cu interfeţe diferite;
* când pe obiectele dintr-o structură trebuie efectuate mai multe operaţii neînrudite;
* nu se „poluează” clasele cu aceste operaţii
* când se definesc des noi operaţii pe structură, dar clasele structurii nu se modifică des;
* dacă o clasă nu are metoda Accept, se poate deriva pentru a o adăuga;
* șablonul este util când nu se pot modifica sursele unor biblioteci sau platforme pentru adăugarea unor noi funcţii.

Are urmatoarele avantaje:

* adăugarea de noi clase ConcreteElement este dificilă;
* șablonul nu este recomandat în timpul dezvoltării unui sistem, când numărul de clase creşte sau clasele se modifică mult;
* poate reduce încapsularea, obligând obiectele vizitate să-şi expună starea internă;
* vizitatorul nu poate accesa câmpurile private, deci trebuie să existe metode publice de acces în obiectele vizitate.

În figura 5 este reprezentă digrama de clase pentru șablonul Visitor.

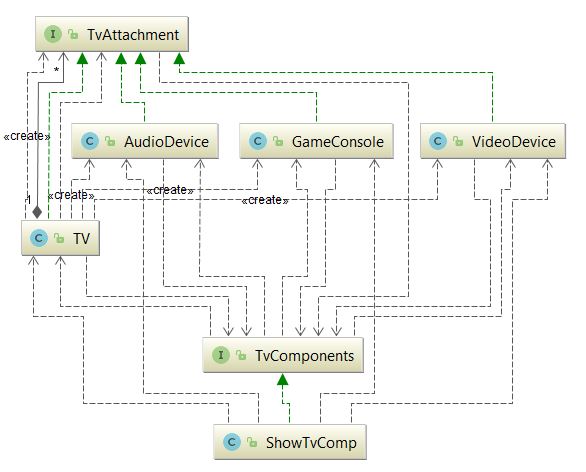


Fig.5 – Visitor Pattern

**Concluzie**

În decursul executării acestei lucrări de laborator am făcut cunoștință cu modul de implementare și utilizare a șabloanelor de comportament. Am implementat următoarele șabloane: Command, Iterator, Mediator, Strategy, Visitor.

**Anexa**

Repozitoriu distant: https://github.com/CornitaConstantin/Lab3IPP