Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatica și Micorelectronică

Catedra Automatica și Tehnologii Informaționale

**Raport**

Lucrarea de laborator nr. 2

# La disciplina: Ingineria Produselor Program

Tema: Șabloane structurale

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

A verifica: lector asistent, Chetrusca Ecaterina

Chișinău 2016

**Scopul și sarcina**

De studiat 5 șabloane structurale  și de implimentat în proiect propriu.

**Noțiuni teoretice**

Sabloanele structurale studiază modelele cu privire la structura sistemului bazat pe clase și obiecte. În acest caz, pot fi folosite următoarele mecanisme:

* Moștenire, atunci când clasa de bază definește interfața iar subclasele – realizarea. Structuri obținute în baza moștenirii se primnesc statice.
* Compoziția, atunci când structurile sunt construite prin combinarea mai multor clase de obiecte. Compoziția produce structuri care pot fi schimbate în timpul.

**Adaptorul (Adapter)**

Acest șablon converteşte interfaţa unei clase în altă interfaţă pe care o aşteaptă clientul. Adaptorul permite să funcţioneze împreună clase care altfel nu ar putea din cauza interfeţelor incompatibile.

**Adaptorul** permite utilizarea unor clase fără a modifica nici codul client nici codul ţintă.

Diagrama de clasa pentru Adapter Pattern

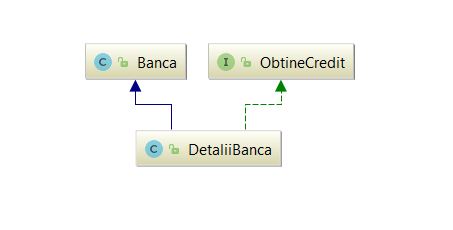


Fig.1 Adapter Patern

**Composite Pattern**

Acest șablon compune obiecte în structuri de tip arbore pentru a reprezenta ierarhii parte-întreg. Şablonul permite clienţilor să trateze uniform obiecte individuale şi compuneri de obiecte.

Pot fi ignorate diferenţele dintre obiectele compuse şi obiectele elementare.

Composite Pattern are 2 responsabilități:

* Gestionarea ierarhiei
* Executarea operaţiilor legate de meniuri

Compromis pentru creşterea transparenţei

Şablonul Composite este uneori utilizat împreună cu şablonul Decorator

Când se utilizează împreună Decoratori şi Compuneri, ei vor avea de obicei o clasă părinte comună.

Decoratorii vor trebui să implementeze interfaţa Component cu operaţii precum Add, Remove şi GetChild.

Diagrama de clasa petru Composite Pattern

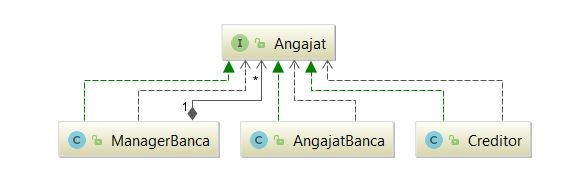


Fig.2 Composite Pattern

**Decorator Pattern**

Sablonul Decorator ataşează responsabilităţi suplimentare unui obiect în mod dinamic. Decoratorii asigură o alternativă flexibilă la derivare pentru extinderea funcţionalităţii.

**Decoratorii** au acelaşi tip ca şi obiectele decorate. Se poate de trimis ca parametru  un obiect decorat în locul obiectului iniţial.

**Decoratorul** foloseşte moştenirea pentru a prelua tipul obiectului decorat şi compunerea pentru a-i schimba comportamentul. Se pot utiliza unul sau mai mulţi decoratori pentru un obiect. Obiectele pot fi decorate dinamic în momentul execuţiei. **Decoratorul** îşi poate adăuga propriul comportament înainte sau după delegare. **Decoratorul** permite adăugarea de noi funcţionalităţi fără modificări în cod. **Decoratorii** pot fi creaţi cu şabloanele **Fabrică** şi **Constructor**

**Digrama de clase pentru șablonul Decorator Pattern**

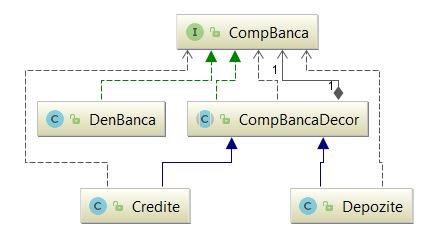


Fig. 3 Decorator Pattern

**Facade Pattern**

***Faţada (Façade)***- asigură o interfaţă unificată la o mulţime de interfeţe dintr-un subsistem. Faţada defineşte o interfaţă de nivel mai înalt care face subsistemul mai uşor de utilizat.

Faţada nu numai că simplifică o interfaţă, ci şi decuplează clientul de subsistemul de componente  
- Subsistemul poate fi de asemenea accesat direct, faţada nu încapsulează subsistemul.

Atât **Faţada** cât şi **Adaptorul** pot împacheta (pot fi wrappere pentru) una sau mai multe clase

* Scopul adapatorului este sa converteaasca interfata;
* Scopul fațadei este sa simplifice interfata.

Diagrama de clasa pentru fatada Pattern

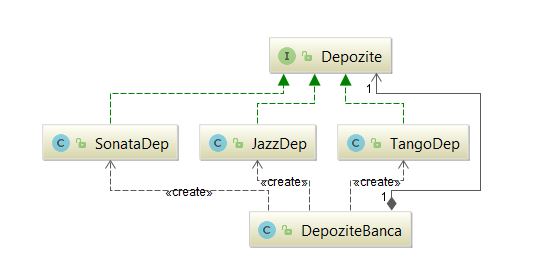


Fig.4 Facade Pattern

**Bridge Pattern**

Sablonul Bridge are o structura similara unui obiect adaptor, dar are un cu totul alt scop: el este destinat separarii unei interfete de implementarea ei, astfel incat cele doua sa poata fi modificate independent, cu usurinta. Scopl unui adaptor este sa schimbe interfata unui obiect existent.

Sablonul Proxy defineste un reprezentant sau un surogat pentru un alt obiect si nu-i modifica interfata.

Decupleaza o abstractizare de implementarea ei, astfel incat cele doua sa poata varia independent.

Aplicabilitate

Sablonul Bridge se utilizeaza cand:

* dorim sa evitam o legatura permanenta intreo abstractizare si implementarea acesteia. Astfel de cazuri ar putea fi, de exemplu, cand implementarea trebuie selectata sau schimbata a executie.
* atat abstractizarile, cat si implementarile acestora trebuie sa fie extensibile prin generarea subclaselor. In acest caz, sablonul Bridge, permite sa combinam abstractizari si implementari diferite si sa le extindem imediat.
* modificarile in implementarea unei abstractizari nu trebuie sa afecteze clientii; cu alte cuvinte, codul client nu trebuie sa fie recompilat.

Digrama de clasa pentru Bridje Pattern

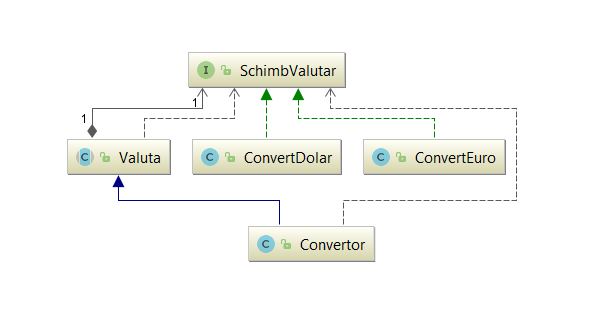


Fig. 5 Bridge Patern

**Rezultatul programului**

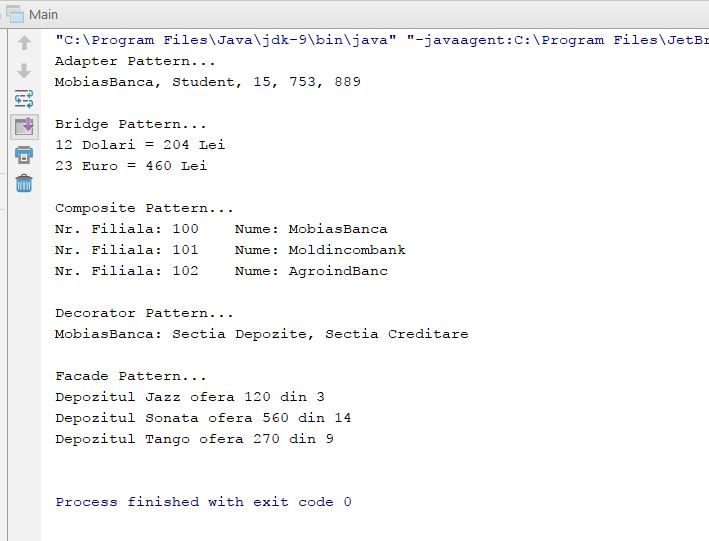


Fig.6 Rezultat

**Concluzie**

Sabloanele structural se ocupa de modul in care sunt compuse clasele si obiectele pentru a forma structurri mai complexe. Sabloanele structurale de clasa folosesc mostenirea pentru a compune interfete sau implementari. Acest tip de sablon este in special util pentru a face sa functioneze impreuna bibliotecile intre clase dezvoltate independent. Un exemplu este forma de clasa Adapter. In general, un adaptor realizeaza o interfata conforma cu o alta, furnizand astfel o abstractizare uniforma pentru interfete diferite. Un adaptor de clasa realizeaza acest lucru prin mostenirea privata de la clasa adaptat. Apoi, adaptorul exprima interfata sa in termenii adaptatului.

.

**Anexa A:**

Repoztoriul distant: https://github.com/CornitaConstantin/Lab2IPP