Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**Lucrare de laborator nr.2**

*Disciplina: Tehnici și mecanisme de proiectare software*

**Tema:** Șabloane de proiectare structurale



**Studentul: gr. TI-206 Raileanu Daniel**

**Coordonator: asist. univ. Buldumac Oleg**

**Chișinău 2023**

**Sarcina:** Să se implementeze la nivel de cod 5 șabloane de proiectare structurale din cele 7.

**Adapter**

**Scopul:** este de a permite colaborarea între două clase care altfel nu ar putea lucra împreună din cauza unor interfețe incompatibile.

**Problema care o rezolvă:** Este necesar să se utilizeze o clasă existentă, care nu este compatibilă cu alte clase sau module din aplicație.

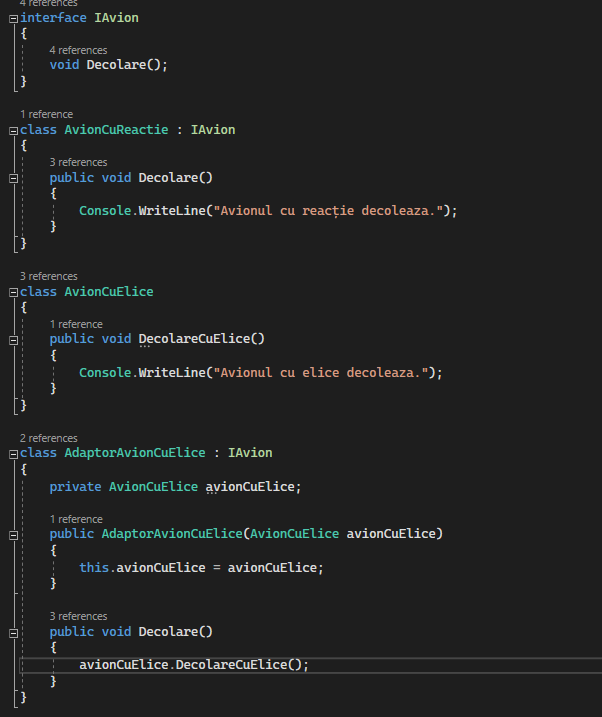


Figura 1 – Implementarea clasei Adapter

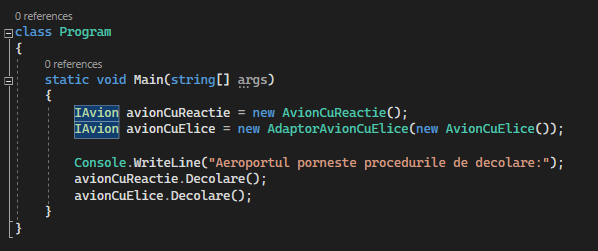


Figura 2 – Crearea/primirea instanței

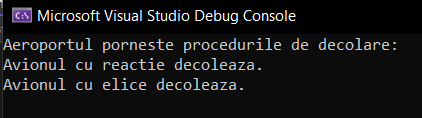


Figura 3 – Executarea șablonului Adapter

**Composite**

**Scopul:** este de a permite manipularea obiectelor individuale și a grupurilor acestora (obiecte compozite) într-un mod uniform.

**Problema care o rezolvă:** este aceea că, în anumite situații, poate fi dificil să se gestioneze o colecție de obiecte individuale și grupuri de obiecte în același timp.

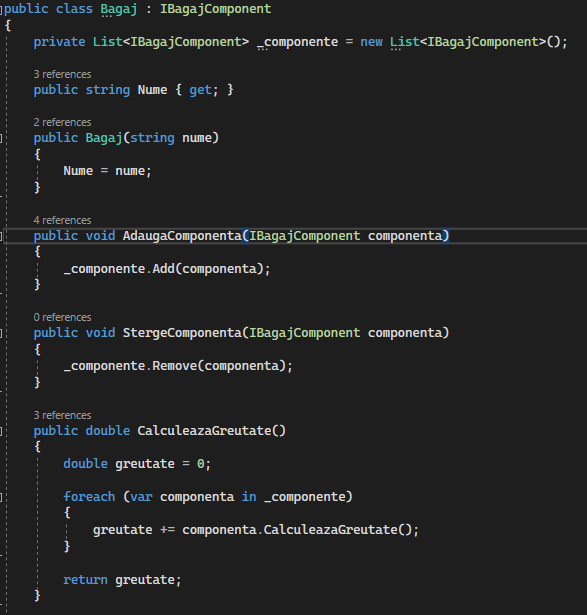


Figura 4 – Executarea șablonului Composite

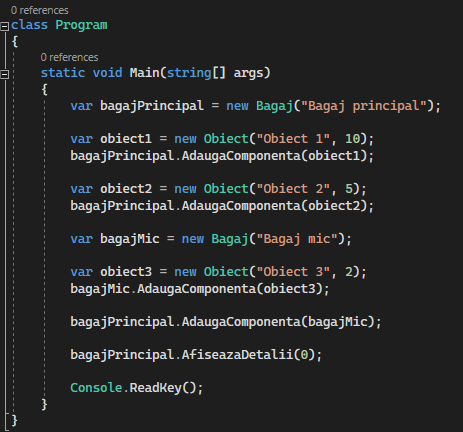


Figura 5 – Crearea/primirea instanței

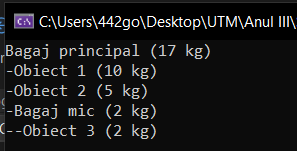


Figura 6 – Executarea șablonului Composite

**Decorator**

**Scopul:** este de a permite adăugarea de funcționalități suplimentare la un obiect existent într-un mod flexibil și extensibil.

**Problema care o rezolvă:** poate fi necesară adăugarea de funcționalități suplimentare la un obiect existent, dar nu se dorește modificarea acestuia în mod direct sau nu este posibilă moștenirea din clasa respectivă.

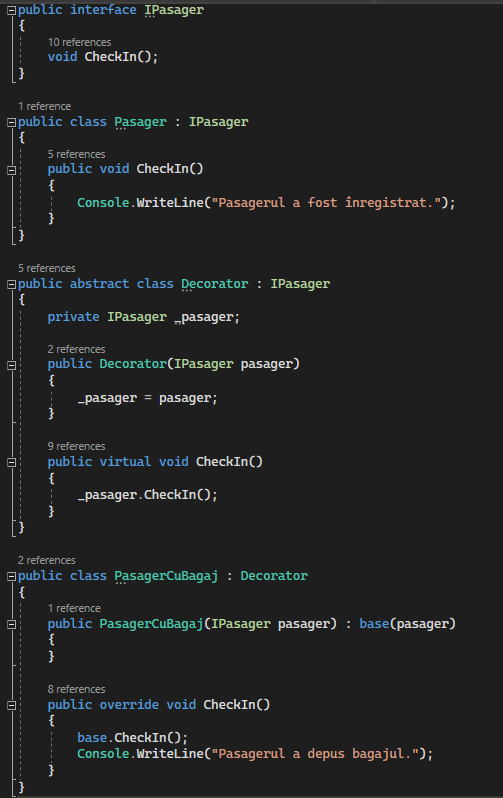


Figura 7 – Executarea șablonului Decorator

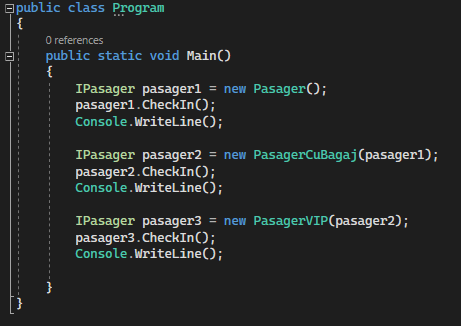


Figura 8 – Crearea/primirea instanței

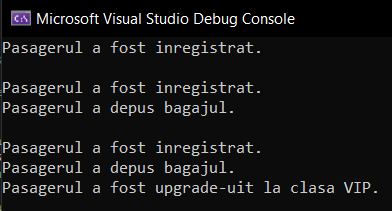


Figura 9 – Executarea șablonului Decorator

**Facade**

**Scopul:** este de a oferi o interfață simplificată și unificată pentru un set complex de clase sau funcționalități, astfel încât acestea să poată fi utilizate mai ușor.

**Problema care o rezolvă:** în anumite situații, poate fi dificil sau complicat să se folosească un set de funcționalități sau clase din cauza complexității lor sau a interdependențelor între acestea.

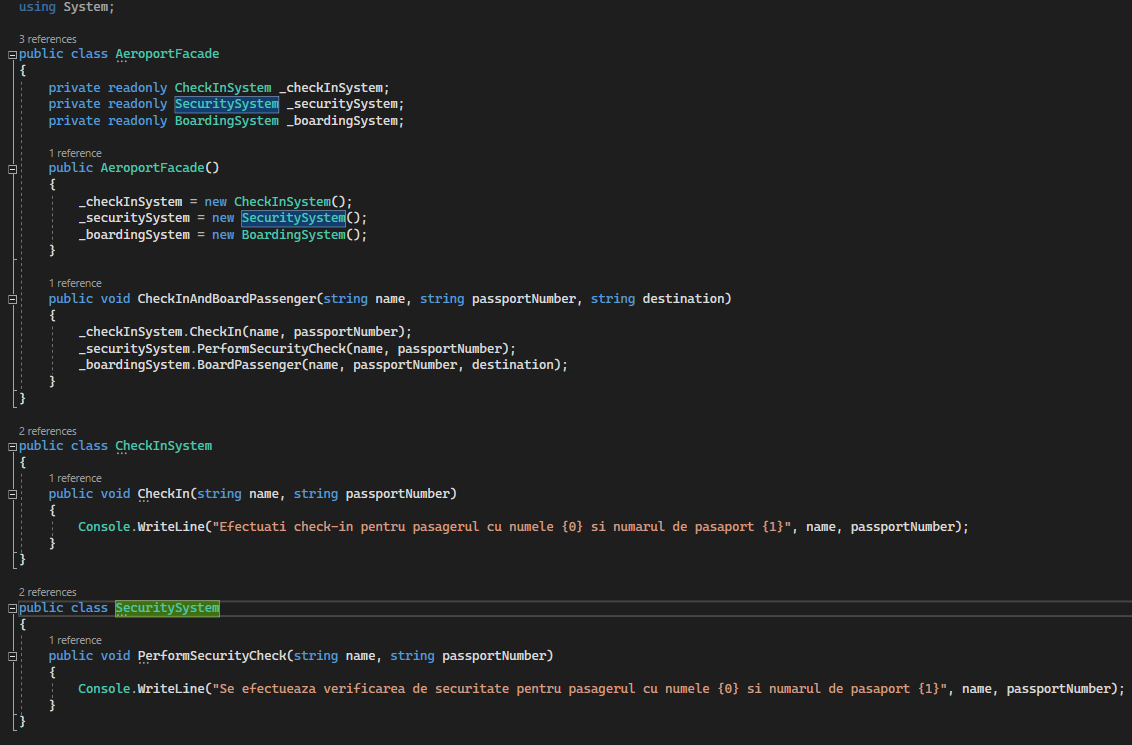


Figura 10 – Executarea șablonului Facade



Figura 11 – Crearea/primirea instanței

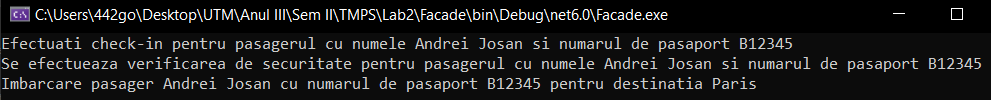


Figura 12 – Executarea șablonului Facade

**Proxy**

**Scopul:** este de a separa crearea obiectelor de utilizarea lor, permițând astfel schimbarea în timp util a implementării obiectelor fără a afecta codul clientului.

**Problema care o rezolvă:** de a permite crearea de obiecte asociate în mod abstract, fără a specifica clase concrete, pentru a reduce cuplarea între obiecte și a crește flexibilitatea și extensibilitatea programului.

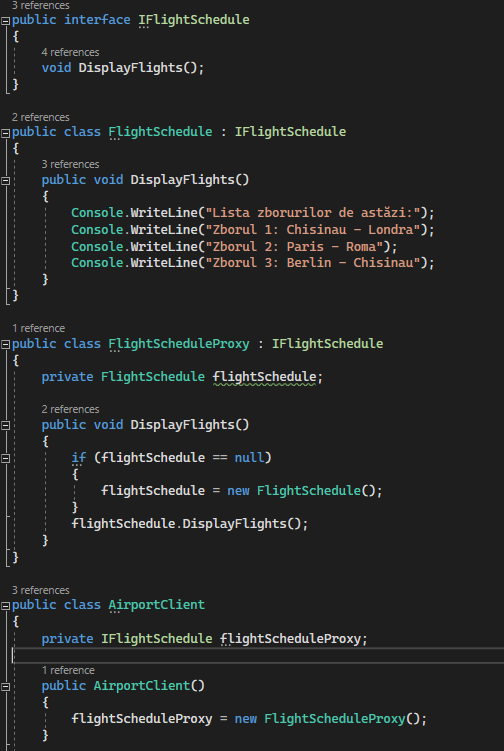


Figura 13 – Executarea șablonului Proxy

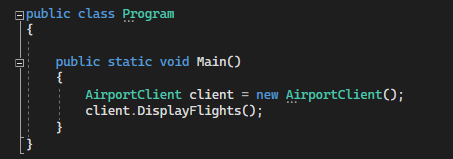


Figura 14 – Crearea/primirea instanței

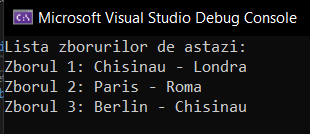


Figura 15 – Executarea șablonului Proxy

**Concluzie**

În urma efectuării lucrării de laborator am învățat despre șabloanele de proiectare și scopul lor în programare. a oferi mecanisme și abstracții pentru a crea obiecte într-un mod flexibil și extensibil, în funcție de necesitățile aplicației. Aceste sabloane se concentrează pe modul în care obiectele sunt create și inițializate, în locul modului în care acestea sunt utilizate și modificate.

Am explorat diferite sabloane creationale, inclusiv Adapter, Composite, Decorator, Facade și Proxy. Fiecare dintre aceste sabloane a fost conceput pentru a rezolva o anumită problemă în dezvoltarea de software.

Adapterul a fost utilizat pentru a adapta interfețe diferite ale claselor existente, astfel încât acestea să poată fi utilizate împreună. Acesta poate fi util atunci când se dorește utilizarea unei clase existente într-un context diferit sau atunci când se dorește evitarea modificării claselor existente.

Composite-ul a fost utilizat pentru a construi structuri ierarhice complexe de obiecte, în timp ce Decorator-ul a fost utilizat pentru a adăuga comportament suplimentar obiectelor existente fără a le modifica direct.

Facade-ul a fost utilizat pentru a oferi o interfață simplificată și unificată pentru un set complex de clase sau funcționalități, astfel încât acestea să poată fi utilizate mai ușor.

Proxy-ul a fost utilizat pentru a oferi un obiect de înlocuire sau o referință pentru un alt obiect, astfel încât acesta să poată fi accesat sau modificat fără a fi necesară o interacțiune directă cu obiectul original.

În concluzie, utilizarea acestor sabloane creationale mi-a permis să creez aplicații mai eficiente, flexibile și ușor de întreținut. Prin intermediul acestor sabloane, am putut să creez obiecte într-un mod flexibil și extensibil, să ascund complexitatea creării și utilizării acestora și să îmbunătățesc modularitatea și flexibilitatea aplicației.

**Bibliografie**

https://github.com/442god/TMPS