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.1**

*Disciplina: Tehnici și mecanisme de proiectare software*

**Tema:** Șabloane de proiectare creaționale



**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 creaționale din cele 6 (cu Object Pool)

**Singleton**

**Scopul:** este utilizat pentru a restricționa numărul de instanțieri ale unei clase la un singur obiect, deci reprezintă o metodă de a folosi o singură instanță a unui obiect în aplicație.

**Problema care o rezolvă:** Modelul Singleton rezolvă două probleme în același timp, încălcând Principiul responsabilității unice

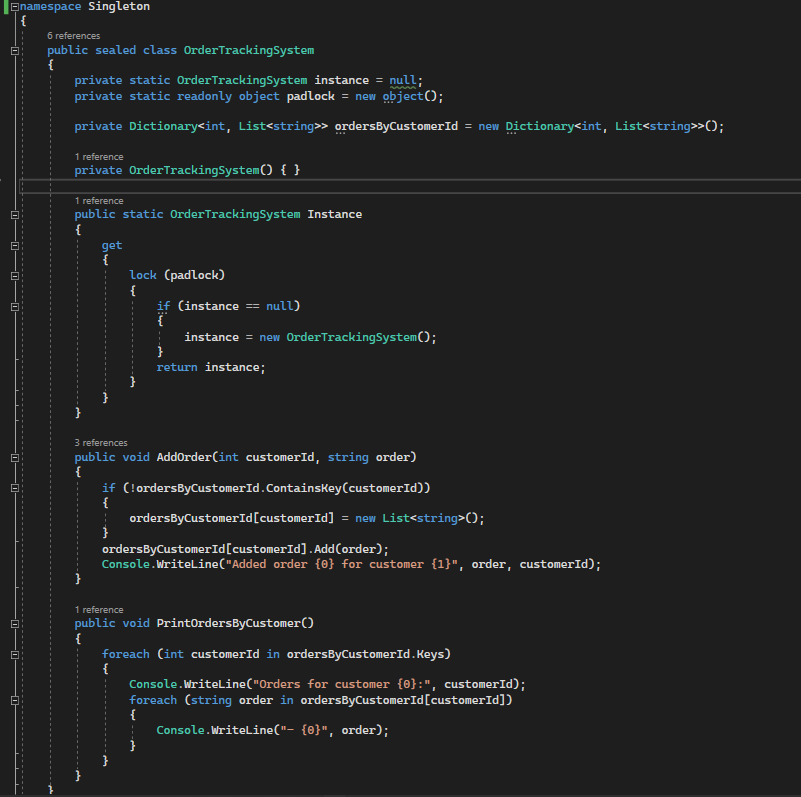


Figura 1 – Implementarea clasei Singleton

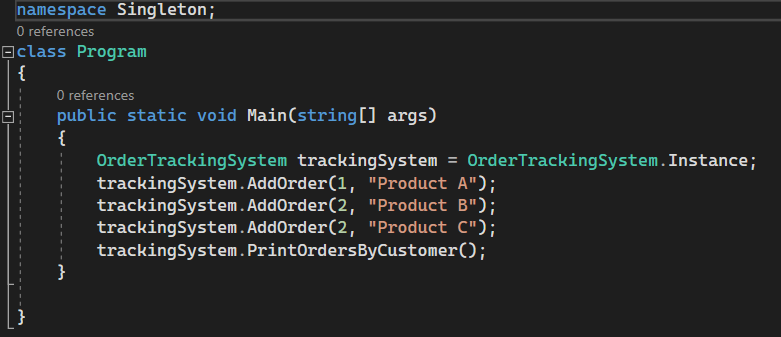


Figura 2 – Crearea/primirea instanței

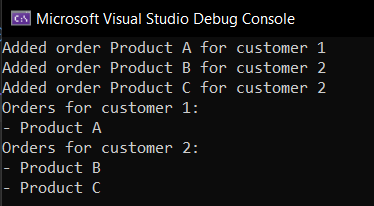


Figura 3 – Executarea șablonului Singleton

**Builder**

**Scopul:** este utilizat pentru a separa construcția complexă a unui obiect de reprezentarea sa și de a permite crearea diferitelor tipuri de obiecte cu aceeași structură de bază.

**Problema care o rezolvă:** de a crea obiecte complexe, care pot avea mulți parametri și opțiuni diferite, astfel încât crearea manuală a obiectului poate fi o sarcină dificilă și propice la erori.

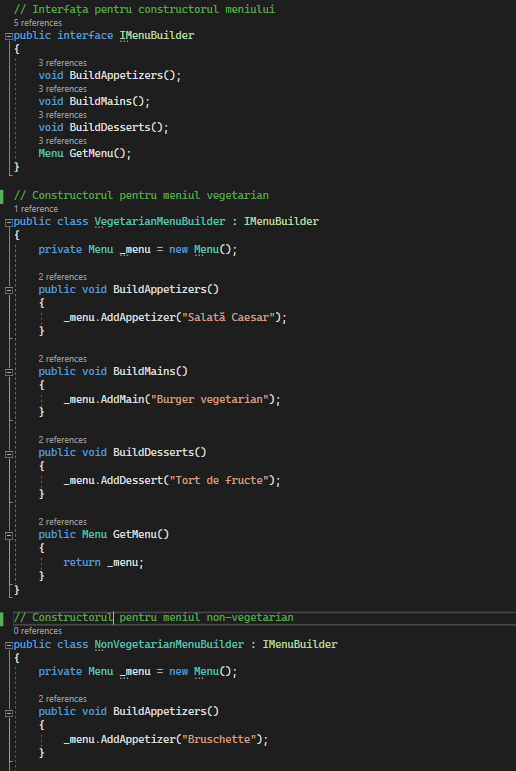


Figura 4 – Executarea șablonului Builder

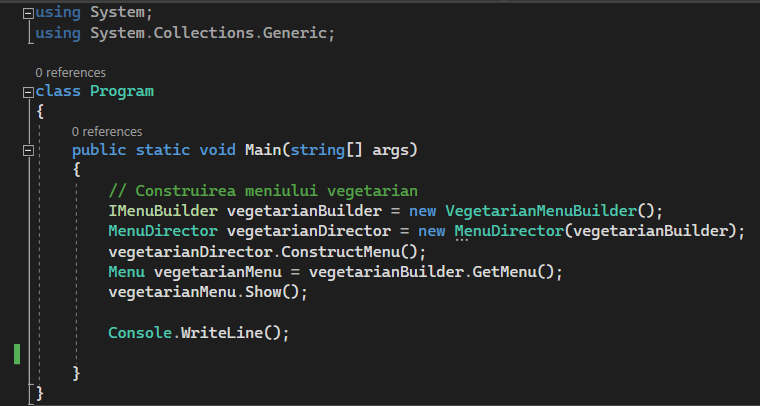


Figura 5 – Crearea/primirea instanței

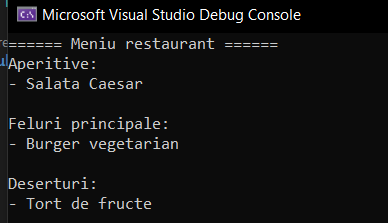


Figura 6 – Executarea șablonului Builder

**Prototype**

**Scopul:** este de a permite crearea de noi obiecte prin clonarea sau copierea obiectelor existente, în locul creării de noi obiecte prin intermediul constructorilor.

**Problema care o rezolvă:** de a reduce timpul și costurile alocate pentru crearea de noi obiecte, în special atunci când obiectele sunt complexe și necesită o configurație specifică.

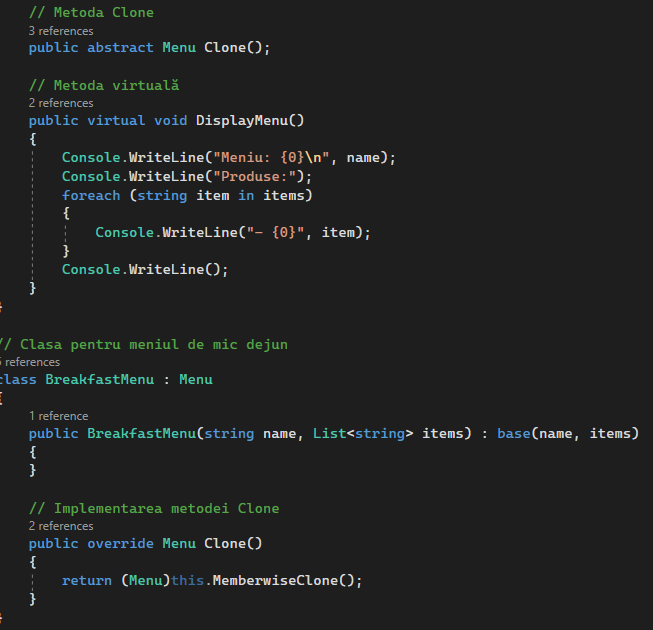


Figura 7 – Executarea șablonului Prototype

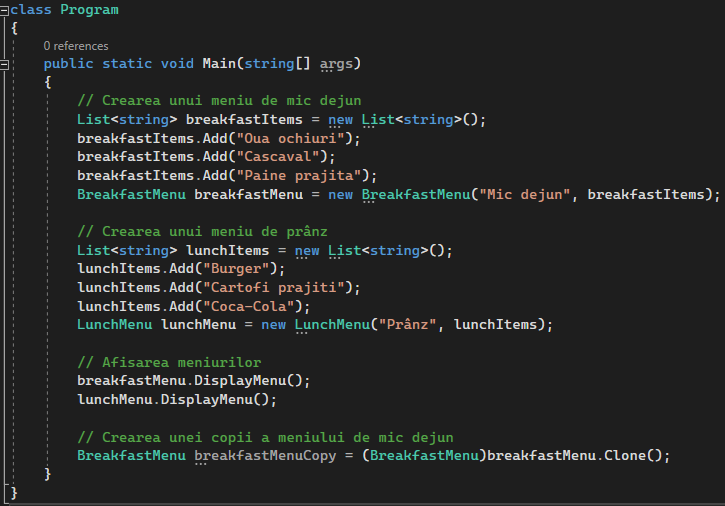


Figura 8 – Crearea/primirea instanței

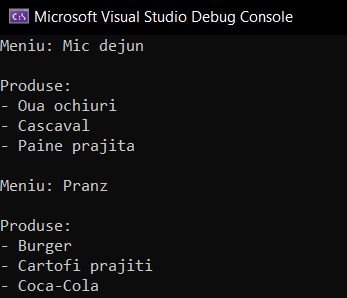


Figura 9 – Executarea șablonului Prototype

**Factory Method**

**Scopul:** este de a defini o interfață pentru crearea obiectelor într-o clasă de bază, dar permite subclaselor să își modifice tipul obiectelor care vor fi create.

**Problema care o rezolvă:** permite unei clase să prea responsabilitatea creării obiectelor către subclase, fără a-i impune constrângeri rigide sau specifice.

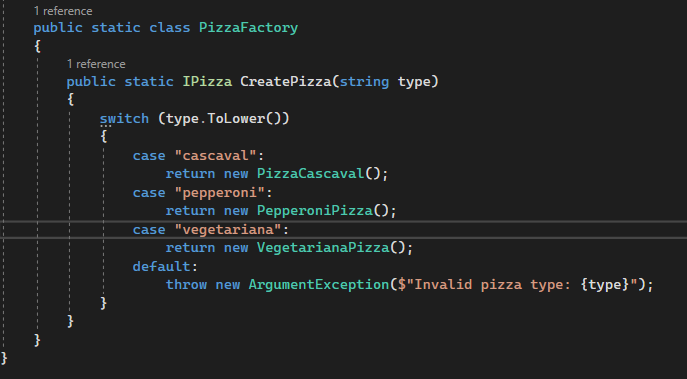


Figura 10 – Executarea șablonului Factory Method

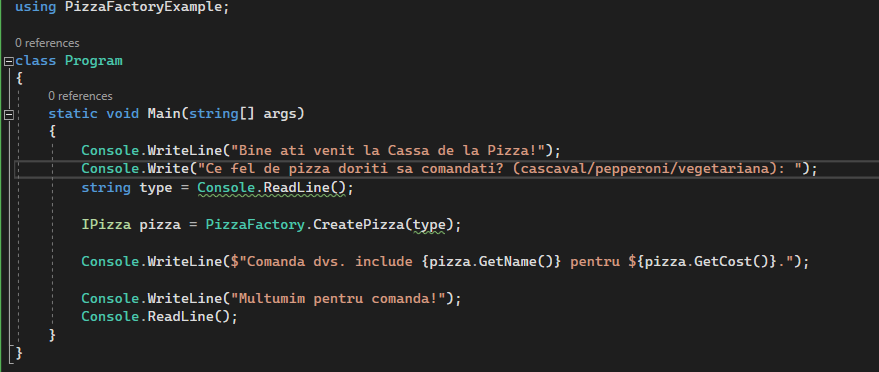


Figura 11 – Crearea/primirea instanței

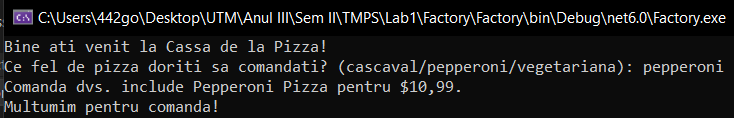


Figura 12 – Executarea șablonului Factory

**Abstract Factory**

**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.

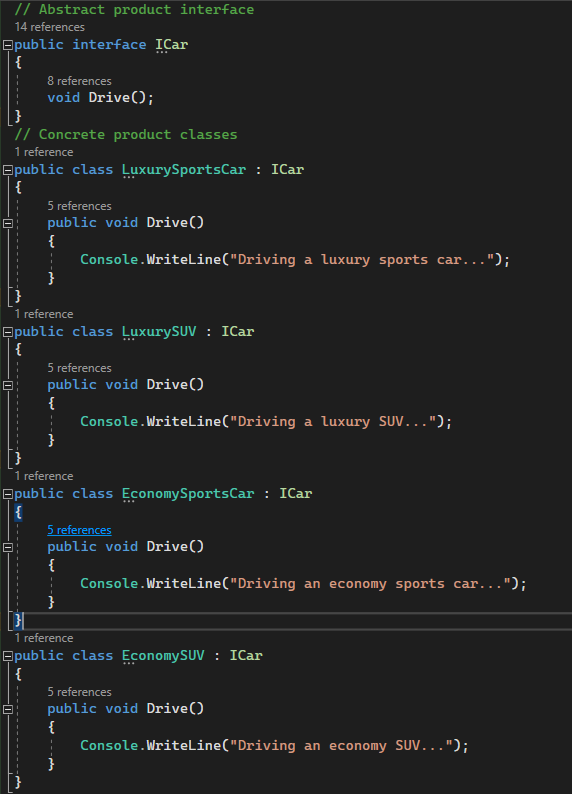


Figura 13 – Executarea șablonului Abstract Factory

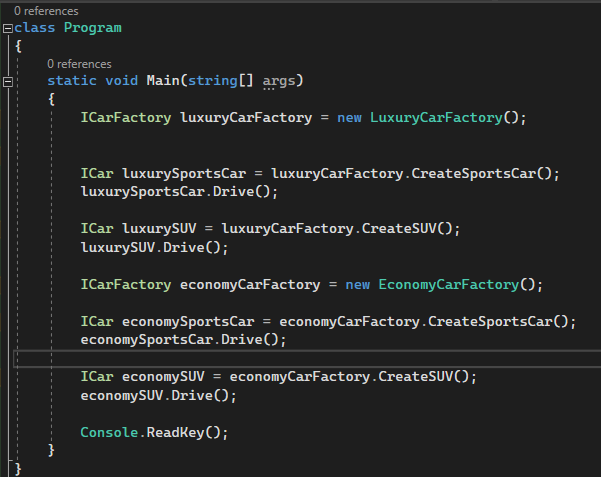


Figura 14 – Crearea/primirea instanței

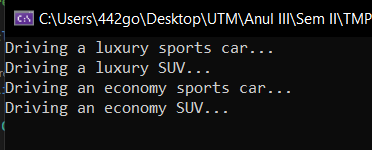


Figura 15 – Executarea șablonului Abstract

**Concluzie**

În urma efectuării lucrării de laborator am învățat despre șabloanele de proiectare și scopul lor în programare. Aceste șabloane au fost concepute pentru a ajuta programatorul să evite erorile și să scrie un cod mai eficient, ușor de întreținut și de extins în viitor.

Primul șablon și care se consideră cel mai ușor este Singleton. Acesta este simplu din cauza că face un singur lucru – asigură o clasă că aceasta are doar o singură instanță. Aceasta este realizată datorită unei metode care returnează instanța clasei, iar dacă clasa a fost deja instanțiată, returneză aceeași instanță.

Al doilea șablon, Builder, este următorul despre care voi vorbi, acesta ne permite să creăm obiecte complexe pas cu pas și în caz că un obiect nu are nevoie de o proprietate pe care noi o avem creată, pur și simplu o omitem.

Urmatorul șablon pe care l-am utilizat în cadrul acestui laborator este șablonul Prototype. Acesta ne permite să clonăm obiecte deja create fără ca obiectul nou creat să depindă de obiectul inițial. Aceasta este făcută pentru aceea ca în caz că o să dorim să schimbăm o valoare a unui câmp din obiectul clonat, obiectul inițial să-și păstreze valoarea inițială.

Un alt sablon de proiectare utilizat este Factory. Acesta propune o interfață pentru a crea un obiect pe baza ei, iar această interfață este comună pentru toate clasele care o implementează. Facotry este utilizată pentru aceea ca logică masivă dintr-un constructor dintr-o clasă să fie mutată în altă parte și o clasă principală să aibă grijă de această logică.

Iar ultimul sablon a fost Abstract Factory. Acesta are aceeași logică de utilizare ca și șablonul Factory despre care am discutat anterior, doar că el creează familii de obiecte.

**Bibliografie**

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