**ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI,**

**FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ**

**PROIECT**

**CALITATE SI TESTARE SOFTWARE**

Aplicatie pentru primirea si livrarea comenzilor din cadrul unei cofetarii

**STUDENT:**

**Ciobanu Cristina-Oana**

**Seria A, Grupa 1064**

Cuprins

[I. Definirea si justificarea pattern-urilor implementate 3](#_Toc451442407)

[II. Definirea si detalierea metodelor testate prin Unit Testing 4](#_Toc451442408)

[III. Definirea si descrierea Test Case-urilor 5](#_Toc451442409)

[IV. Definirea si descrierea Test Suite-ului 6](#_Toc451442410)

[V. Descrierea sumara a functiilor aplicatiei cu referire la pattern-uri si la metodele testate 6](#_Toc451442411)

[Bibliografie 7](#_Toc451442412)

# Definirea si justificarea pattern-urilor implementate

In cadrul aplicatiei prezentate, a fost nevoie de implementarea a 7 design pattern-uri:

* Singleton
* Factory
* Strategy
* Command
* Observer
* Chain of responsabilities
* Façade

**Singleton**-face parte din categoria design pattern-urilor creationale. Acesta este utilizat pentru a restrictiona numarul de instante ale obiectelor dintr-o anumita clasa la un singur obiect.

Implementare: Singleton presupune existenta unei metode statice care returneaza instanta obiectului. Daca obiectul nu exista, se creeaza, in cazul in care obiectul exista se returneaza referinta la acesta. Constructorul trebuie sa fie privat, obiectele de tipul clasei singleton se obtin doar prin intermediul metodei.

Acest pattern a fost implementat in vederea construirii un singur obiect de tip cofetarie. Exista o singura cofetarie.

**Factory**- face parte din categoria design pattern-urilor creationale, fiind util in cazul in care se doreste crearea unor obiecte dintr-o familie de obiecte, fara a expune logica crearii obiectelor, nu sunt necesare cunostinte cu privire la structura interna a acestora.

Implementare: Avem nevoie de o interfata comuna(ITort) sau o clasa abstracta, ce va fi implementata de clasele concrete (TortFrisca, TortCiocolata, TortFructe); de o clasa Factory prin intermediul careia se vor crea obiectele de tipul dorit. Clasa Factory contine o metoda getTort() care returneaza tortul dorit, in functie de parametrul setat functiei.

**Facade-** face parte din categoria design pattern-urilor structurale. Acesta are rolul de a ascunde complexitatea sistemului, punand la dispozitia utilizatorului o interfata simplificata, mai usor de utilizat.

**Chain of responsabilities sau lant al responsabilitatilor-** este un design pattern comportamental, utilizat in cazul in care evenimentul poate fi tratat de mai multi “subiecti”, fiecare subiect fiind capabil sa gestioneze evenimentul in anumite conditii, in cazul in care un subiect nu poate trata evenimentul in cauza, acesta este trimis succesorului, daca acesta exista.

Implementare:

In aplicatia avuta in vedere, design pattern-ul chain of responsabilities a fost implementat pentru a gestiona procesul de livrare.

Avem nevoie de o clasa abstracta(Curier) ce contine o metoda abstracta care va gestiona evenimentul si clasele concrete(Motociclist si SoferMasina) care extind clasa abstracta si suprascriu metoda din clasa abstracta.

**Command-** este un design pattern comportamental, folositor atunci cand se doreste gestionarea unor cereri. Cererile sunt adunate si trimise catre clasa insarcinata cu rezolvarea lor.

Acest design pattern a fost implementat pentru adunarea tuturor produselor pe care clientul le doreste, produse ce sunt trimise mai apoi spre preparare cofetarului.

Implementare:

Clasa OperatorComenzi este clasa insarcinata cu adunarea tuturor comenzilor ce vor fi trimise mai apoi spre preparare cofetarului. Clasa contine un obiect de tip Client si lista de comenzi.

Clasa ComandaTort contine o referinta catre un obiect de tip cofetar si implementeaza implementeaza interfata IComanda ce contine metoda prepara().

**Strategy-**  face parte din categoria design pattern-urilor comportamentale si este util in cazul in care se doreste implementarea unor algoritmi diferiti in functie de strategie.

Acest design pattern a fost implementat pentru stabilirea modului in care se va face plata comenzii.

Implementare:

Avem nevoie de o interfata IPlataComanda si clasele concrete(PlataCard si PlataRamburs); clasa EfectuarePlata ce contine o referinta la un obiect de tip IPlataComanda si metoda de efectueazaPlata ce primeste ca parametru un obiect de tip IPlata comanda. In functie de tipul de plata setat se va apela metoda din clasa respectiva.

**Observer**- este un design pattern comportamental util in cazul in care se doreste notificarea unor anumite componente la producerea unui eveniment.

Am implementat acest pattern pentru notificarea clientului in momentul in care comanda lui este finalizata.

Implementare:

Avem nevoie de o clasa abstraca pentru gestionarea tipurilor diferite de observatori(IObserver) si clasa concreta Client ce implementeaza interfata IObserver.

Interfata INotificare ce contine metodele de adaugare, stergere observatori si de anuntare si clasa ComandaPlasata care implementeaza interfata INotificare.

# Definirea si detalierea metodelor testate prin Unit Testing

-setNume()-Metoda seteaza clientului numele trimis ca parametru. Utilizand un bloc try-catch am testat daca metoda trateaza cazul in numele clientului este null. Testul a demonstrat ca metoda testeaza daca valoarea introdusa este null

-setAdresa()-Metoda seteaza clientului valoarea trimisa ca parametru .Utilizand un bloc try-catch am testat daca metoda trateaza cazul in adresa clientului este null. Testul a demonstrat ca metoda testeaza daca valoarea introdusa este null

-getInstance()- Metoda returneaza o singura instanta a clasei respective, aceasta este creata o singura data, dar poate fi referita de mai multe ori. Utilizand metodele assertSame si assertEquals am verificat daca metoda getINstance returneaza aceeasi instanta sau creeaza obiecte noi.

-getTort()-Metoda creeaza un obiect din familia torturilor, pe baza parametrului setat. Aceasta returneaza un obiect de tip ITort. Folosind metoda asserTrue() am testat daca metoda functioneaza corect si returneaza un obiect de tipul clasei concrete.

# Definirea si descrierea Test Case-urilor

Test Case-ul Test\_Chain- contine unit teste pentru testarea metodelor din clasele folosite pentru implementarea acestul design pattern.

Test Case-ul Test\_Client- contine unit teste pentru testarea metodelor din clasa Client

Test Case-ul Test\_ComandaPlasata-contine unit teste pentru testarea metodelor din clasa ComandaPlasata

Test Case-ul Test\_EfectuarePlata-contine unit teste pentru testarea metodelor din clasa EfectuarePlata

Test Case-ul Test\_Facade-contine unit teste pentru testarea metodelor din clasa Facade

Test Case-ul Test\_Factory-contine unit teste pentru testarea metodelor folosite pentru implementarea design pattern-ului factory

Test Case-ul Test\_OperatorComenzi-contine unit teste pentru testarea metodelor din clasa OperatorComenzi

Test Case-ul Test\_Singleton-contine unit teste pentru testarea metodelor folosite pentru implementarea design pattern-ului singleton

# Definirea si descrierea Test Suite-ului

Test Suite-ul TestSuiteCommandPattern-contine test case-urile folosite pentru testarea acestui pattern

Test Suite-ul TestSuiteStrategy-contine test case-urile folosite pentru testarea acestulipattern

Test Suite-ul TestSuiteToateTestele-contine toate test case-uri din proiect.

# Descrierea sumara a functiilor aplicatiei cu referire la pattern-uri si la metodele testate

O cofetarie(design pattern-ul singleton) primeste comenzi pentru diferite tipuri de torturi, obiecte din familia tort( design pattern-ul factory), livrandu-le la domiciliul clientului. Cofetaria realizeaza trei mari tipuri de torturi(cu frisca, cu ciocolata sau cu fructe).

Comenzile sunt preluate de catre un operator, care are rolul de a trimite comenzile catre cofetar(design pattern-ul command). Clientul are la dispozitie doua modalitati de plata(cu cardul sau cash) (design pattern-ul strategy). Dupa preluarea comenzii clientul este notificat cu privire la valoarea totala a comenzii si alte informatii de interes pentru acesta(design pattern-ul observer).

Pentru livrarea comenzilor, cofetaria dispune atat de masini cat si de motociclete. Pentru comenzile nu foarte voluminoase livrarea se realizeaza cu o motocicleta, in caz contrar cu o masina.(design pattern-ul chain of responsabilities)

Pentru simplificarea procesului de preluare si livrare a comenzilor am folosit design pattern-ul facade.

# Bibliografie

<http://acs.ase.ro/Media/Default/documents/cts/curs/Curs%20CTS%20%20Design%20Patterns%202016.pdf>

<http://www.tutorialspoint.com//design_pattern/index.htm>

<http://acs.ase.ro/software-quality-testing>