Academia de Studii Economice din București

Facultatea de Cibernetică, Statistică și Informatică economică

Calitate software și testare

Student:

Pârvu Florin

Grupa 1053

Profesor coordonator:

Boja Cătălin

CUPRINS

[Definirea si detalierea pattern-urilor utilizate 3](#_Toc420596147)

[Singleton 3](#_Toc420596148)

[Factory Method 3](#_Toc420596149)

[Facade 4](#_Toc420596150)

[Metodeletestate prin Unit Testing 4](#_Toc420596151)

[Test Case 4](#_Toc420596152)

[Test Suite 5](#_Toc420596153)

[Descrierea aplicației 5](#_Toc420596154)

[Bibliografie 5](#_Toc420596155)

# Definirea si detalierea pattern-urilor utilizate

Un design pattern (șablon software) este soluție care poate fi aplicată pentru un anumit tip de probleme, pentru care codul folosit în urmă cu ceva timp a fost eficient. Design pattern-ul va fi refolosit pentru probleme curente de același tip. Design pattern-urilor sunt soluții testate care permit reutilizarea codurilor sursă și arhitecturilor și înțelegerea mai facilă a acestora.

Design pattern-urile au ca scop rezolvarea problemelor de design pentru cazuri particulare. Orice design pattern este caracterizat de nume (identificare unică), problemă (definește scopul și contextul), soluție (pseudo-cod ce descriere elemente și relațiile dintre ele) și efecte (avantajele și dezavantajele utilizării). În funcție de scopul pentru care este folosit un design pattern poate fi creațional, structural sau comportamental.

Design patterns creaționale sunt: Abstract Factory, Builder, Factory Method, Prototype și Singleton.

Design patterns structutale sunt: Adapter, Bridege, Composite, Decorator, Facade, Flyweight și Proxy.

Design patterns de comportament sunt: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State și Strategy.

## Singleton

Singleton reprezintă o tehnică prin care se limitează numărul de instanțe ale unei clase la una singura. Componentele acestui șablon sunt: un constructor privat (apelabil doar din clasa), un atribut static, privat, de tipul clasei ce reprezintă instanța unică și o metodă publică ce dă acces la instanța unică

În aplicație, am implementat clasa Singleton folosind design pattern-ul Singletone ,deoarece gestionează conexiunea la o baza e date folosind o singură instanță.

## Factory Method

Factory Method defineşte o interfaţă pentru crearea unui obiect, dar lasă subclasele să decidă care clasă va fi instanţiată. Componentele acestui șablon sunt: interfața ce definește tipurile generice de obiecte ce pot fi create, clasa concretă ce definește tipul de obiecte ce poate fi creat, clasa abstractă ce definește interfața unui generator de obiecte și clasa concretă ce implementează generatorul de obiecte. Cu acest șablon pot fi generate obiecte noi care aparțin aceleiași familii și care au în comun interfața prin care sunt gestionate unitar, noi tipuri din aceeși familie putând fi adăugate fără modificări.

## Facade

Facade simplifică interfața unui modul/subsistem. Soluția conține o mulțime de clase iar execuția unei funcții presupune apeluri multiple de metode aflate in aceste clase. Clasele nu se modifica însă se construiește un layer intermediar ce permite apelul/gestiunea facila a metodelor din mai multe interfețe. Utilă in situația În care framework-ul creste in complexitate si nu este posibila rescrierea lui pentru simplificare. Apelurile către multiplele interfețe sunt mascate de aceasta interfață comună. Se adaugă doar un layer intermediar ce ascunde complexitatea framework-ului din spate.

În aplicație am folosit clasa Facade pentru a reduce complexitatea sistemului prin asigurarea unei interfaţe comună unei mulţimi de interfeţe dintr-un subsistem.

# Metodeletestate prin Unit Testing

Metodele folosite pentru rularea testeler se gasesc in clasele Carte si Casa: succesiuneLitere(), Taxe(), Promotii(), Total(), impartireProfit(), Plata(), Depozit().

# Test Case

Test Case-urile folosite pentru clasa Casa:

Test\_suma\_casa – testare suma corecta din casa

Test\_impartire\_profit\_corect – testare impartire profit in mod corect

Test\_impartire\_profit\_eronat – testare impartire profit in mod incorect

Test\_zecimale\_depozit – testare dacă s-au folosit mai mult de doua zecimale

Test\_valori\_limita - test valori care depasesc limita superioară

Test\_valori\_nule - testare valori nule

Test Case-urile folosite pentru clasa Carte:

TestCarteSetarePret – testare daca setter-ul seteaza pretul corect

TestCarteTitlu – testare daca titlul contine o succesuine de litere

TestCarteTitluNull – testare daca titlul carti este NULL

TestCarteTitluReturnat – testare daca numele asteptat este cel furnizat de catre constructor

Test automat

TestCasaAutomat - Test automat citire valori din fisier

# Test Suite

În aplicație am folosit două Test Suite, unul pentru testelor aplicate clasei Carte și al doilea pentru testele aplicate clasei Casa.

# Descrierea aplicației

Aplicația gestionează produsele unei edituri. Prin Singleton putem crea o singură instantă pentru o baza de date. Prin Factory Method am definit o interfaţă pentru crearea unui obiect, iar subclasele pot să decidă care clasă va fi instanţiată. În urma utilizării șablonului Facade am obținut reducerea complexităţii sistemului.

# Bibliografie

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