**ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI,**

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

**DOCUMENTATIE**

**PROIECT CALITATE SI TESTARE SOFTWARE**

**STUDENT:**

**Cartianu Ileana-Cristina**

**Seria A, Grupa 1064**

**CUPRINS**

PATTERN-URI IMPLEMENTATE ..................................................................................... 3

METODE TESTATE PRIN UNIT TESTING ...................................................................... 3

DESCRIERE TEST CASE-URI ............................................................................................ 4

DESCRIERE TEST SUITE ................................................................................................... 5

DESCRIERE FUNCTII APLICATIE (cu referire la patterns si metodele testate) ............... 5

**PATEERN-URI IMPLEMENTATE**

1. Singleton = proiectul administreaza actiuni ale unei pensiuni (instanta unica);
2. Factory = tipul camerelor;
3. Builder = tipul rezervarii;
4. Strategy = clientul solicita rezervarea prin email sau telefonic;
5. Observer = atunci cand clientul face o rezervare, el este adaugat la newsletterul pensiunii;
6. Chain of responsibility = daca clientul solicita un discount, receptionerul poate accepta doar daca acesta este mai mic de 15 procente. Altfel, este nevoit sa paseze responsibilitatea catre manager;
7. Composite = includerea alimentelor in meniu si a meniurilor specific in meniul general;
8. Façade = simplificarea procesului de preluare comanda si realizare nota de plata.

**METODE TESTATE PRIN UNIT TESTING**

1. getters = pentru toate clasele realizate s-a verificat daca la apelarea functiei de getCaracteristica este returnat rezultatul asteptat;
2. notadeplata () = s-a verificat atat normal, cat si prin cross check, daca totalul pe care un client trebuie sa il plateasca este acelasi ca in scenariul implementat in test;
3. size () = dupa popularea unui array am verificat daca dimensiunea acestuia este cea asteptata, adica daca s-a populat precum a fost asteptat;
4. add () = in cazul meniurile am verificat daca inregistrarile (alimentele/cei care sunt abonati la newsletter) sunt introduse conform specificatiilor;
5. remove () = in acelasi timp am verificat daca inregistrarile (alimentele/cei care sunt abonati la newsletter) sunt sterse, cand este cazul;
6. descriere () = pentru aceasta metoda am testat daca atunci cand meniul este gol, programul arunca o exceptie in loc sa afiseze continutul meniului (pentru ca nu ar avea ce sa afiseze);
7. notifica (mesaj) = atunci cand nu avem nici un abonat, este aruncata o exceptie. Nu putem transmite un mesaj catre nici un abonat;
8. getters singletone = am creat 2 obiecte initializate diferit. Am verificat daca atributele primului obiect sunt egale cu atributele celui de-al doilea obiect;
9. test singleton = am comparat 2 obiecte de tip singleton, initializate cu valori diferite;
10. inchiriereCapacitateTotala (zile, discount) = am verificat atat normal, cat si prin cross check care ar fi pretul inchirierii capacitatii totale;
11. getVarsta () = in cazul clientilor am folosit si un test de valoare extrema si am testat daca este apelata o exceptie in cazul in care varsta este negative;
12. teste extreme = am realizat teste extreme si pentru perioada si nr de camere corespunzatoare unei rezervari, considerand ca nu se pot realiza rezervari pentru mai putin de o camera sau mai putin de o zi, dar si pentru discount, care nu poate fi mai mic decat 0;
13. teste inverse = am realizat diferite teste inverse de aflare a unui pret sau a numarului de camera, a perioadei, samd;

**DESCRIERE TEST CASE-URI**

1. TesteComenzi = test case in cadrul caruia am verificat daca obiectele de tip aliment sunt create corespunzator, daca nota de plata este realizata in mod corect, daca toate alimentele comandate sunt incarcate intr-un array;
2. TesteMeniu = test case in cadrul caruia am verificat daca obiectele de tip meniu sunt create corespunzator, daca este aruncata o exceptie cand meniul este gol, daca meniul este populat corespunzator cu alimente si daca se sterge in mod corect un aliment cand se urmareste acest lucru;
3. TesteNewsletter = test case in cadrul caruia am verificat daca luna in care se trimite newsletterul este corecta, daca adaugarea si eliminarea abonatilor din lista de followers se realizeaza in mod corespunzator si daca este aruncata o exceptie atunci cand se doreste trimiterea unui newsletter desi lista de abinati este goala;
4. TestePensiune = test care in cadrul caruia am verificat daca al doilea obiect creat este egal cu primul obiect, in cazul singleton creandu-se o singura instanta de obiect indiferent de numarul de apeluri;
5. TesteRezervari = test case in cadrul caruia am verificat daca este aruncata o exceptie atunci cand un client este initializat cu o varsta negativa, si de asemenea daca perioada, numarul de camere sau discountul nu respecta cerintele urmarite. Am verificat aceste lucruri prin teste extreme. De asemenea am verificat metodele de set din cadrul pattern-ului *BUILDER* si verificaDiscount din cadrul pattern-ului *CHAIN OF RESPONSIBILITY*.

**DESCRIERE TEST SUITE**

In cadrul test suite-ului, denumit GeneralTestSuite am apelat toate test case-urile mentionate mai sus.

Suita de teste este reprezentata de rularea impreuna a mai multor test case-uri. In Junit, pentru a realiza aceasta actiune folosim atat @RunWith cat si @Suite.

**DESCRIERE FUNCTII APLICATIE**

Aplicatia isi propune sa reproduca actiunile folosite in cadrul unei pensiuni, precum solicitari de rezervare, comenzi din meniu, solicitari de discount si chiar interactiune cu clientii prin intermediul unui newsletter.

Am folosit *SINGLETON* pentru a crea pensiunea, deoarece este o instanta unica. Mai departe am folosit *FACTORY* pentru a crea obiecte de tipul camerelor, deoarece toate sunt din aceeasi familie, diferentele fiind foarte mici.

Urmatorul pattern folosit este *BUILDER* pentru realizarea rezervarilor. Am folosit acest pattern pentru ca am urmarit crearea de obiecte complexe intr-un mod structurat, dar si pentru ca detaliile unei rezervari se pot schimba la orice moment si acest pattern este cel mai potrivit pentru schimbarea rapida a caracteristicilor unui obiect de tip rezervare, in cazul nostru. Am folosit *STRATEGY* pentru a stabili daca rezervarea a fost facuta telefonic sau prin email.

Mai departe, pentru interactiunea cu clientii am realizat un newsletter, iar pentru aceasta actiune am folosit patternul *OBSERVER*. Al saselea pattern implementat este reprezentat de *CHAIN OF RESPONSIBILITY*, cu ajutorul caruia se doreste sa se trateze situatia in care este solicitat un discount. In cazul in care discountul este mai mic de 15%, receptionerul poate lua singur decizia de a il accepta. In caz contrar, este nevoit sa paseze responsabilitatea catre manager.

Am flosit *COMPOSITE* pentru a popula meniul de bauturi si meniul de mancare cu alimente, mai apoi populand meniul general cu aceste doua meniuri. Nu in ultimul rand am folosit *FAÇADE* pentru a simplifica tot procesul unei comenzi: Clientul comanda anumite alimente, ospatarul ia comanda si realizeaza si nota de plata.