Academia de Studii Economice

Facultatea de Cibernetică, Statistică şi Informatică Economică

**Proiect la disciplina**

***Calitate și Testare Software***

 

Studentă: Vătășoiu Roxana-Ioana

Grupa 1053

IE, Seria B

**CUPRINS**

1. **Definirea și detalierea design pattern-urilor utilizate ..................................................3**
   1. Singleton......................................................................................................3
   2. Builder.........................................................................................................3
   3. Template.......................................................................................................4
2. **Definirea și detalierea metodelor testate prin Unit Testing.......................................4**
3. **Definirea și descrierea Test Case-urilor.................................................................6**
4. **Definirea și descrierea Test Suite-ului.....................................................................6**
5. **Descrierea sumară a funcțiilor aplicației...............................................................6**

**Bibliografie...................................................................................................................7**

1. **Definirea și detalierea design pattern-urilor utilizate**
   1. ***Singleton***

Singleton este un pattern creațional. Astfel, prin utilizarea sa se crează o singură instanță (unică). Instanțace poate fi creată o singură data, dar poate fi referită de mai multe ori. Acesta asigură un singur punct de acces, vizibil global, la respectiva instant.

Clasa Client îndeplinește rolul de singleton. Aplicația informatică are la bază principiul conform căruia nu poate exista mai mult de un client pe sesiune.

Client este persoana care dorește să se înregistreze la un hotel, adică să închirieze o cameră sau mai multe, în funcție de necesitățile sale. Acesta deține un anumit buget în care va trebuie să se încadreze în momentul închirierii unei camera. De asemenea, el va putea beneficia de un discount pentru o sumă achitată mai mare.

* 1. ***Builder***

Builder-ul reprezintă un pattern din categoria celor creaționale, alături de Singleton și Factory. Are ca scop crearea în mod structurat (incremental) de obiecte complexe printr-un mecanism independent de procesul de realizare. Clientul construiește obiectele complexe specificând doar tipul si valoarea sa, fără a cunoaște detaliile interne ale obiectului (cum stochează si reprezintă valorile). Procesul de construire a obiectelor poate fi utilizat pentru a defini obiecte diferite din aceeași familie. Algoritmul de creare a obiectului este flexibil, deoarece clientul alege ce părți să fie create.

Clasa Cameră este un pattern de tip Builder, care se caracterizează printr-un număr al acesteia, etajul la care se află, numărul de paturi și un preț. Pe lângă aceste atribute, camerele diferă între ele prin facilitățile oferite: televizor, frigider, aer condiționat. Astfel, vom putea define obiecte distincte prin setarea anumitor atribute.

* 1. ***Template***

Template este un pattern comportamental, care încapsulează un algoritm ai cărui pași depind de o clasă derivată. Implementarea unui astfel de algoritm presupune o secvență predefinită și fixă de pași. Se caracterizeză prin prezența unei metode în care se găsesc apelate metode într-o anumită ordine. Implementează principiul Hollywood: "Don't call us, we'll call you.“.

În aplicația prezentă, am exemplificat acest pattern prin definirea unor metode care resprezintă etapele parcurse de un client pentru a face o rezervare la un hotel: acesta va specifica câte camere rezervă, va introduce datele personale, va selecta modul de plată și va plăti suma respectivă.

1. **Definirea și detalierea metodelor testate prin Unit Testing**
   1. ***rezervăCameră()***

Pentru această metodă s-a realizat testul *testRezervăCameră(),* folosit în cazul în care o cameră este nulă.

* 1. ***ștergeCameră()***

În această situație, a fost efectuat testul denumit *testStergereCameră(),* utilizat atunci când se dorește a fi ștearsă o cameră nulă.

* 1. ***pozițieCameră()***

Metoda pozițieCameră() are la bază următoarele teste:

* testNegativIndexCameră(), utilizat în cazul în care indexul introdus este o valoare negativă;
* testExtraIndexCameră(), folosit când este introdusă o valoare mai mare decât lungimea listei de camere.
  1. ***getSize()***

Această metodă presupune efectuarea unui test – *testZeroDimensiuneListăCameră(),* care realizează verificarea listei în cazul în care este vidă.

* 1. ***înregistrareCameră()***

Pentru metoda de mai sus au fost efectuate următoarele teste:

* testZeroValoareIdCameră(), în cazul în care a fost introdus ca valoare numărul zero pentru cameră;
* testNegativValoareIdCameră(), atunci când numărul camerei a fost trimis ca valoare negativă;
* testMaxValoareIdCameră(), utilizat când a fost introdusă o valoare prea mare, aberantă pentru numărul unei camere.
  1. ***plăteșteCameră()***

În cazul acestei metode, testele folosite au fost:

* testMaxValoareBuget(), necesar pentru situația în care prețul unei camere este mai mare decât bugetul disponibil al clientului;
* testNegativValoarePreț(), atunci când valoare prețului este una negativă;
* testZeroValoarePreț(), utilizat în momentul în care valoarea introdusă pentru prețul camerei este 0;
* testInvalidValoare(), folosit pentru valori aberante ale prețului, precum 0.0005 RON.
  1. ***primeșteDiscount()***

Această metodă se bazează pe eficiența testelor de mai jos:

* testZeroCost(), utilizat la introducerea unei valori de 0 pentru atributul cost;
* testNegativCost(), în cazul unei valori negative pentru variabila cost, deoarece nicio cameră nu poate avea un preț negativ;
* testInvalidCost(), folosit pentru valorile limită, care nu au sens în acest caz, precum 0.0005 RON.
  1. ***validareDuratăCazare()***

Pentru această metodă s-au utilizat următoarele trei teste:

* testZeroDuratăCazare(), folosit pentru introducerea unui număr de zile de cazare egal cu 0;
* testNegativDuratăCazare(), util în testarea prezenței unei valori negative pentru numărul de zile;
* testInvalidDuratăCazare(), folosit în situația în care există o valoare mult prea mare a numărului de zile.

1. **Definirea și descrierea Test Case-urilor**

Referitor la Test Case-uri, am implementat următoarele: TestClient și TestSuiteClient. În timp ce în TestSuiteClient contează ordinea în care apelăm metodele, în TestClient se pune accentul pe ordinea în care le construim.

Astfel, în primul test găsim totalitatea testelor definite mai sus, iar în cel de al doilea este implementat un test suite.

1. **Definirea și descrierea Test Suite-ului**

Test Suite-ul este implementat în clasa TestClientSuite. În interiorul acestuia am introdus patru metode: testRezervareCameră(), testZeroDuratăCazare(), testNegativDuratăCazare(), testInvalidDuratăCazare(). Acesta este cazul în care are loc adăugarea parțială de teste, dar există și o adăugare totală a tuturor testelor din clasa TestClient.

1. **Descrierea sumară a funcțiilor aplicației**

Funcțiile definite în cadrul clasei Client sunt următoarele:

* *rezervăCameră():* această metodă este apelată atunci când respectivul client dorește rezervarea unei camere.
* *ștergeCameră():* funcția este apelată în momentul în care se va șterge o cameră, presupune anularea unei rezervări pentru o anumită cameră.
* *pozițieCameră():* această metodă se ocupă de afișarea anumitor informații despre o cameră care face parte din listă, de la o poziție specificată.
* *getSize():* rolul funcției este acela de a returna lungimea listei de camere.
* *înregistrareCameră():* metoda este utilă în momentul rezervării unei camere, în sensul testării validității numărului camerei.
* *plăteșteCameră()*: prin intermediul acestei funcții este calculat un cost asociat prețului camerei.
* *primeșteDiscount():* funcția calculează valoarea care trebuie achitată în cazul aplicării discount-ului.
* *validareDuratăCazare():* metoda aceasta afișează numărul de zile de cazare ale clientului.

**Bibliografie**

* *suportul de curs*