Proiect

Calitate si Testare Software

Pandra Alexandra Denisa

[Grupa 1065, Seria A]

1. ***Definirea si justificarea pattern-urilor implementate:***

Singleton

Este un design pattern creational prin intermediul caruia se creeaza o singura instanta, care poate fi insa referita de mai multe ori. De asemenea, asigura gestionarea unei resurse sau au unui eveniment in mod centralizat.

Se implementeaza printr-un constructor privat, un atribut static, privat, de tipul clasei ce reprezina instanta unica si utlizeaza o metoda publica ce da acces la instanta unica.

Am utilizat singleton-ul pentru clasa „Gestiune”, care cauta, actualizeaza prin citirea din fisier si afiseaza detalii despre biletele disponibile pentru anumite evenimente.

Astfel, resursele sunt gestionate centralizat printr-o instanta unica.

Simple Factory

Obiectele au in comun interfata, nu sunt create prin constructori ci prin metoda de tip factory. Avantajul utilizarii acestui design pattern consta in faptul ca pot fi adaugate noi tipuri, din aceeasi familie fara a face modificari. Am implementat simple factory pentru a putea adauga cu usurinta noi tipuri de tipuri de evenimente precum „Competitie sportiva”, „Festival”.

Facade

Facade este utilizat atunci cand solutia este complexa, se bazeaza pe multe clase, iar executia unei functii presupune apeluri multilple ale unor metode aflate in aceste clase. Clasele nu sunt modificate, insa se construieste un layer intermediar pentru a facilita utilizeaza metodelor.

Pentru intocmierea unei facturi, se utilieaza clasele „Bilet”, „BiletComandat” si „Factura”. Pentru a gestiona mai usor apelul tuturor metodelor, am creat un design pattern de tip facade.

Strategy

Permite alegerea la run-time a agloritmului care va fi utilizat pentru procesarea unor date. Algortimul poate fi alea pe baza unor conditii descrise la executie in functie de contextul datelor de intrare.

Am implementat strategy pentru alegerea unei locatii, in functie de tipul evenimentului.

1. ***Definirea si detalierea metodelor testate prin Unit Testing***

Unit Testing este o metoda simpla si rapdia de testare a codului sursa , ce are loc in faza de dezvoltare.

Am testat metodele : *assertEquals(message,expected, actual), assertNotNull(object), assertTrue(condition),assertFalse(condition).*

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

Test Case-ul este o clasa ce defineste setul de obiecte pentru a rula mai multe teste. In clase ce extinde TestCase se definesc unit teste ce folosesc metodele *assert* pentru a evalua modul de executie a metodelor din clase testata.

Proiectul „GestionareEvenimente” cuprinde urmatoarele TestCase-uri:

*TestareFactura, TestareGestiune, TestareOrganizatori.*

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

Test Suite-ul reprezinta o colectie de cazuri de testare (test cases) ce vor fi evaluate impreuna.

1. ***Descrierea sumara a functiilor aplicatiei cu referire la pattern-uri si la metodele testate***

Functiile utilizate pentru testare :

Din clasa Factura – *getTotalFaraTaxe(), getTotalCuTaxe(), getTaxe(), getTotalCateogrieCuTaxe(), numaraEvenimente()*

Din clasa Gestiune – *cautaBilet()*

Din clasa Organizatori - *diponibilitateLocatie(), overtime()*