Proiect Calitate si Testare Software

Cabinet Veterinar

Isaila Adriana

Grupa 1070

Seria B

1. Definirea si justificarea pattern-urilor implementate

Pentru realizarea proiectului, s-au folosit urmatoarele designe pattern-uri:

* Singleton
* Builder
* Simple Factory
* Façade
* Observer
* Prototype
* State
* Template

**Singleton** a fost folosit in scopul crearii unui singur obiect de tip Cabinet, deoarece, activitatea este gestionata pe actiunea si personalul unui singur obiect de acest tip, si in niciun moment al rularii programului, nu mai este necesar de un alt obiect identic.

*Metode utilizate:*

* Cabinet()
* getInstance()
* howMessage()
* angajeazaAngajat(Angajat angajatNou)
* concediazaAngajat(Angajat angajatNou)

**Builder** a fost folosit deoarece permite instantierea unui obiect,fara ca acesta sa contina toate atributele. Astfel, se poate decide care sunt atributele ce vor fi afisate prin implementarea constructorului. Pentru ca exista posibilitatea ca utilizatorul sa isi doreasca sa afiseze si altceva pe langa atributele din constructor, se construiesc interfetele publice get si set pentru fiecare atribut.

*Metode utilizate:*



**Simple Factory** este patternul ce permite crearea unei familii de obiecte. In cazul nostrum, ne dorim sa cream familia Angajat. Prin intermediul designe-ului, este permisa crearea a doua obiecte diferite, Medic si Asistent dar care implementeaza aceeasi interfata, Angajat, cee ace inseamna ca ambele implementeaza aceeasi metoda da in feluri diferite.

*Metode utilizate:*



Am folosit pattern-ul **Façade**, deoarece acesta permite ascunderea complexitatii sistemului si ofera o interfata mai usoara si placuta utilizatorului. Prin intermediul acestui tip de designe, in interiorul clasei FacadePregatireOperatie sunt definite cateva metode care sunt necesare unui medic atunci cand acesta doreste sa execute o operatie. Avand in vedere toate aceste lucruri, atunci cand un medic intra intr-o operatie, el poate instantia direct un obiect de tipul FacadePregatireOperatie fara sa fie nevoie sa scrie codul de fiecare data pentru actiunile pe care trebuie sa le indeplineasca.

*Metode utilizate:*



**Observer** a fost implementat deoarece un client are obtiunea de a face o programare la medic pentru animalul sau prin intermediul unui apel telefonic. In momentul in care s-a stabilit o programare, toti asistentii primesc o notificare.

Prin intermediul designe pattern-ului **Prototype**, se permite crearea unei interfete ce creaza o clona a obiectului curent. In ceea ce priveste implementarea lui in proiect, acesta a fost folosit cu scopul de a crea o copie pentru aparatura folosita in operatie, in cazul in care aceasta de foloseste de mai multe ori(sa nu se creeze de fiecare data o alta aparatura, deoarece acest lucru ar fi costisitor ca timp).

*Metode utilizate:*



**State** este designe-ul ce permite hotararea in ceea ce priveste actiunea ce urmeaza sa se desfasoare, la run time. In proiect este implementat cu scopul de a oferi optiunea unui client de a isi face o programare sau de a se prezenta direct la urgente cu animalul.

*Metode utilizate:*



S-a implementat designe-ul **Template** pentru a putea define cu exactitate modul de executare al metodelor.

*Metode utilizate:*



1. Definirea si detalierea metodelor testate prin Unit Testing

Au fost testate urmatoarele metode:

* Testarea metodi CalculeazaSalariu din clasa Medic in functie de fiecare caz: varsta, vechime, salariu.
* Testarea metodei calculeazaCostOperatie in diferite circumstante, din clasa Pisica();
* Testarea metodei verificaAnimal() din clasa PrototipAparatOperatie.
* Testare metodei adaugaRezervare() din clasa Receptioner.
* Testarea Handler-ului pentru a putea fi efectuata o programare si a actiunii de validare a programarii(programarea a fost inregistrata in sistem)
* Testarea metodelor de calculare a zilelor libere pe care le are un angajat
* Testarea tipului de angajat pe baza clasei DefinesteActiune.

1. Definirea si descrierea Test Case-urilor

* TestAnuntat contine doua metode:
* testProgramareHandlerRezerva() este metoda prin care se testeaza daca programarea este facuta pe baza unui handler existent sau nu. Pentru ca programul sa functioneze, handlerul trebuie sa fie diferit de null.
* testProgramareNull() testeaza daca programarea a fost inregistrata in sistem. Daca rezultatul este null, atunci va fi apelata metoda fail("Metoda nu arunca exceptie pe null"), prin care se anunta faptul ca nu a fost lansata o exceptie pentru acest caz.
* TestAsistent:
* testCalculeazaZileLibereZileLucrateNegative() verifica daca numarul de zile lucrate este negativ.
* testCalculeazaZileLibereZileLucratePreaMulte() calculeaza daca numarul de zile lucrate este prea mare.
* testCalculeazaZileLibereZileLucrateValoriBune() calculeaza daca numarul de zile lucrate contine valorile potrivite(nu este un numar prea mare, dar nici unul negativ)
* TestDefinesteActiune:
* testGetAngajatMedic() verifica daca actiunea intoarce un rezultat de tipul Medic.
* testGetAngajatAsistent()verifica daca actiunea intoarce un rezultat de tipul Asistent.
* testGetAngajatReceptioner()verifica daca actiunea intoarce un rezultat de tipul Receptioner.
* testGetAngajatFemeiServici() verifica daca actiunea intoarce un rezultat de tipul null, deoarece in clasa DefinesteActiune(), nu a fost tratat cazul pentru angajarea unei femei de serviuciu.
* testGetAngajatNull() verifica daca actiunea intoarce un rezultat de tipul null.
* TestMedic:
* testCalculeazaSalariuVarstaSubValoareaMinima() testeaza daca varsta este sub valoarea minima, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuVarstaNegativa()testeaza daca varsta este negativa, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuVarstaZero()testeaza daca varsta este egala cu 0, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuVarstaPreaMare()testeaza daca varsta este prea mare, si daca da, dar nu este aruncata o exceptie pentru acest caz, este apelata functia fail().
* testCalculeazaSalariuValoriNOrmale() testeaza daca valorile varstei, ale vechimii si salariului vechi sunt valori valide, si returneaza rezultatul dupa formula.
* testCalculeazaSalariuVechimeNegativa()testeaza daca vechimea este negativa, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuVechimeZero()testeaza daca vechimea este 0, adica medical a fost de abia angajat, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuVechimePreaMica()testeaza daca vechimea este prea mica, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuVechimePreaMare()testeaza daca vechimea este prea mare,si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuSalariuMediuNegativa()testeaza daca valoarea salariului vechi este negativa, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuSalariuMediuZero()testeaza daca valoarea salariului vechi este 0, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuSalariuMediuPreaMic()testeaza daca valoarea salariului vechi este prea mic, si daca da, daca este aruncata o exceptie.
* testCalculeazaSalariuSalariuMediuPreaMare()testeaza daca valoarea salariului vechi este prea mare, si daca da, daca este aruncata o exceptie.
* TestOperatiePisica este testul care verifica care ar fi costul unei operatii pentru o pisica, in functie de anumite criterii. Mai exact, se testeaza metoda calculeazaCostOperatie() din clasa Pisica.
* public void testcalculeazaCostOperatieVarstaZero()
* testcalculeazaCostOperatieVarstaPreaMare()
* testcalculeazaCostOperatieGreutateNulla()
* testcalculeazaCostOperatieGreutatePreaMare()
* testcalculeazaCostOperatieValoriNOrmale()testeaza daca valorile introduse sunt normale si daca rezultatul returnat respecta formula.
* TestPrototipAparatOperatie verifica daca poate sa fie folosit in operatie laser-ul in functie de animal. Doar cainele si pisica pot fi operate cu aceasta aparatura.
* testVerificaAnimalCaine()
* testVerificaAnimalPisica()
* testVerificaAnimalZebra()
* TestReceptioner verifica daca rezervarea este facuta in mod corect; cu alte cuvinte daca au fost introduse datele, si nu sunt lasate campurile cu valori nule si daca rezervarea a fost facuta in trecut sau pentru o perioada viitoare.
* testAdaugaRezervareEmptyString()
* testAdaugaRezervareNullString
* testAdaugaRezervareInsuficienteCaractere
* testAdaugaRezervarePreaMulteCaractere()
* testAdaugaRezervareLimitaInferioaraString()
* testAdaugaRezervareLimitaSuperioaraString()
* testAdaugaRezervareValoriNormale()
* testAdaugaRezervareDataInTrecut()
* testAdaugaRezervareInAceiasZi()

1. Definirea si descrierea Test Suite-ului

Test Suite-ul contine toate testele de mai sus:

* TestReceptioner
* TestAnuntat
* TestAsistent
* TestDefinesteActiune
* TestMedic
* TestOperatiePisica
* TestPrototipAparatOperatie

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

Se gestioneaza activitatea intr-un cabinet veterinar. In cadrul acestuia, se pot face angajari. Tipul angajatilor este: medic, receptioner si asistent. Cand un client suna sa isi faca o programare, toti asistentii sunt anuntati de aceasta. La latitudinea clientului ramane si care sunt servicile la care el apeleaza: fie isi face programare, fie solicita interventie de urgenta asupra animalului.