**Academia de Studii Economice din București**

**Facultatea de Cibernetică, Statistică si Informatică Economică**

**CALITATE SI TESTARE SOFTWARE**

Coordonator științific:

Conf. univ. dr. Catalin BOJA

Student:

Alexandru NISTOR

**Bucureşti**

**2015**

**CUPRINS**

*Capitolul 1.* **Definirea si detalierea pattern-urilor utilizate....................................................... 3**

*Capitolul 2.* **Definirea si detalierea metodelor testate prin Unit Testing ................................. 4**

*Capitolul 3 .* **Definirea si descrierea Test Case-urilor ............................................................... 5**

*Capitolul 4 .* **Definirea si descrierea Test Suite-urilor ............................................................... 6**

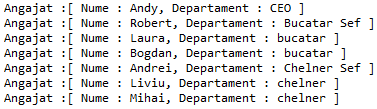
*Capitolul 5 .* **Descrierea sumara a functiilor aplicatiei cu referire la pattern-uri si la metodele testate ...................................................................................................... 7**

*Bibliografie* ..................................................................................................................................16

**CAPITOLUL 1**

**Definirea si detalierea pattern-urilor utilizate**

**Acest proiect este construit in vederea gestiunii unui restaurant. In restaurant lucreaza angajati ce au diferite functii. Pentru o buna intelegere a ierarhiei angajatilor am decis sa utilizez ca si pattern, *composite.*** Astfel codul este simplu de scris deoarece angajatii din ierarhie ii voi trata unitar.



Ulterior am decis sa adaug anumite atribute (calitati) angajatilor pentru ca fiecare dintre acestia sa stie ce indatoriri au in restaurant. Pentru acest lucru am decis sa folosesc ***builder****.* Astfel pot seta anumite sarcini pentru fiecare angajat (sa conduca invitatii la masa cand acestia sosec in restaurant, sa aduca nota de plata la sfarsitul mesei, sa curete masa dupa ce invitatii au plecat, etc).

C:\Users\Andy\Desktop\Print2.png

Tot pentru o mai buna gestiune a restaurantului si pentru ca managerul sa stie pe ce oameni se poate baza si in ce moment am aplicat pattern-ul ***factory*** pentru a crea angajati cu disponibilitate de munca de 8 si 4 ore astfel incat, in cazul in care in restaurant au loc evenimente private si mai este nevoie de oameni pentru a gestiona activitatea, managerul sa trimita mesaj catre angajatii care nu sunt in acea tura, pentru a-i anunta ca are nevoie de ei pentru a servi la mese sau pentru a prepara mancarea.

C:\Users\Andy\Desktop\Print3.png

**CAPITOLUL 2**

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

Am considerat ca, daca restaurantul are clienti, pe viitor va fi nevoie de mai mai multi angajati astfel incat am luat in calcul ca la angajare acestia trebuie sa comunice *numarul de telefon* (la care pot fi deranjati in caz ca are loc un eveniment privat), *codul numeric personal* si o *adresa de e-mail*.

Prima data am testat functia *isCorrectCNP* din clasa *Angajat* din pachetul *Patterns.*

Am luat in calcul ca prima cifra sa fie 1, 2, 5, 6 pentru ca testele sa treaca si ca numarul de cifre din cod sa nu fie diferit de 13.

Functia isCorrectNrTel din clasa *Angajat* din pachetul *Patterns* testeaza daca numarul de telefon are formatul corect: numarul de telefon sa aiba 10 cifre, sa inceapa de fiecare data cu 0 si sa nu contina litere in componenta sa.

Functia isCorrectEmail se gaseste tot in clasa Angajat din pachetul Patterns si testeaza formatul adresei de e-mail. Pentru aceasta functie m-am folosit de: StringTokenizer; regex.Matcher, regex.Pattern pentru a delimita prin punct si pentru a verifica un string daca este de forma pattern-ului definit.

Functiile de marire si de taiere salariu se aplica in anumite conditii, care daca nu sunt indeplinite se arunca exceptiile definite in program.

Functia de marire salariu se aplica daca angajatul nu a ajuns la pragul de 3000 de lei, in caz contrar, acestuia ii putem mari salariul cu 1 milion. Functia de taiere salariu se aplica pana cand angajatul ajunge la pragul de 600 de lei salariul. Aceste functii se gasesc tot in clasa *Angajat*  iar in test case am testat atat pentru valori normale cat si pentru valori mai mari sau mai mici (dupa caz).

La angajare am considerat ca fiecare angajat ar trebui adaugat intr-un departament. In aceasta privinta in clasa *Departament* din pachetul *Patterns* mi-am definit o lista de angajati in care am implementat metodele de adaugare, stergere si cautare. Aceste metode sunt testate in test case-ul *TestLista* din pachetul *Tests*.

Atat timp cat exista angajati in firma, trebuie platiti si am creat clasa cont bancar ce reprezinta un cont in care angajatul primeste banii si ii poate retrage. In aceasta clasa am creat metodele *setUp* si *tearDown* si am testat functiile de primire si retragere.

**CAPITOLUL 3**

**Definirea si descrierea Test Case-urilor**

Toate test case-urile se pot regasi in program in pachetul *Tests*.

In Test Case-ul *TestCNP* am testat functia de verificare a formatului de cod. Am testat atat pentru valori normale cat si pentru valori aberante (cu litere, CNP cu mai multe sau cu mai putine caractere). Tot in acest Test Case am utilizat date de test preluate din fisier.

In Test Case-ul *TestNrTelefon* am testat functia de verificare a formatului numarului de telefon. Testul consta in faptul de a verifica functia daca returneaza *true* atunci cand valorile sunt bune sau *false* cand sunt gresite. Am verificat pentru un numar corect, pentru numar cu mai multe si mai putine caractere, pentru numar cu litere in componenta si numar „format doar din litere” etc.

In Test Case-ul *TestEmail* am testat functia de verificare a formatului adresei de e-mail. Mi-am definit un pattern si am verificat daca ce stringul dat de mine e conform cu patternu-ul definit. Astfel am verificat pentru adrese de e-mail cu doua caractere „@”, cu doua „ .com”, cu alt caracter dupa „@” in afara de „ . ” , cu adresa fara domeniu etc.

In Test Case-ul *TestMarireTaiereSalariu* am testat functiile de marire si de taiere a salariului unui angajat. Am testat atat pentru valori normale cat si pentru exceptii cum ar fi: daca angajatul ajunge la un prag de 3000 de lei nu i se mai poate mari salariul iar daca ajunge la pragul de 600 de lei nu i se mai poate miscora salariul, managerul fiind nevoit sa il dea afara de le locul de munca.

In Test Case-ul *TestContBancar* am testat functiile de adaugare bani in cont si de retragere iar in *TestLista* am verificat functiile de inserare, stergere si cautare intr-o lista.

In Test Case-ul *TestMock* testam un obiect de tip Angajat. Un obiect mock este un inlocuitor al obiectului real si e un testing pattern.

**CAPITOLUL 4**

**Definirea si descrierea Test Suite-urilor**

O suita de teste este o colectie de cazuri de testare, cazuri care sunt destinate a fi utilizate pentru a testa un program software pentru a demonstra ca are un anumit set specific de comportamente.

La fel ca la seminar, am decis sa creez doua Test Suite-uri (suite de teste), unul cu toate Test Case-urile si altul cu unele teste alese de mine.

Astfel in pachetul *Tests* gasim testul *TestSuiteAllTest* in care se afla toate testele din program iar in *TestSuiteSomeTests* avem selectate doar anumite Test Case-uri.

**CAPITOLUL 5**

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

Despre metodele testate in Test Case-uri am vorbit in capitolele 2 si 3.

Referitor la functiile aplicate pentru realizarea pattern-urilor le voi explica pe rand.

**COMPOSITE**

**-** Am construit o lista de angajati pe care am implementat doua operatii: de adaugare si de eliminare plus o functie care sa imi intoarca lista de angajati. In clasa de *Test* care contine si void main() imi creez angajati cu diferite functii dupa care stabilesc ierarhia.

**BUILDER**

**-** Am creat clasa *AtributiiAngajat* in care mi-am definit un constructor fara parametrii si gett-eri si sett-eri pentru atributele din clasa.

- In clasa *AtributiiAngajatBuilder* am creat un obiect de tipul clasei *AtributiiAngajat* si am creat sett-eri pentru a seta „calitatile” fiecarui angajat

- Am creat interfata *IAtributiiAngajatBuilder* in care am definit metoda build() pentru ca in void main dupa ce stabilesc „calitatile” angajatilor sa imi construiasca un obiect de tip *AtributiiAngajat* cu acele „calitati” setate de mine.

**FACTORY**

- Am creat o interfata *IAngajat* cu functia *oreSuplimentare*().

- Am facut un *enum* in care am declarat e angajatii cu 4 si 8 ore.

- In clasele *AngajatCu4Ore* si *AngajatCu8Ore* am definit un constructor si metoda *oreSuplimentare*() care trebuie implementata deoarece clasele extind clasa Angajat.

- In clasa *AngajatFactory* am facut o functie care imi intoarce un angajat in functie de tipul acestuia (daca lucreaza 4 sau 8 ore).

- In *void* *main*() imi creez un *AngajatFactory* dupa care doi angajati de tip *Angajat* care ii voi egala cu angajatul de tip *AngajatFactory* si voi accesa *getAngajat* care va lua ca parametru un an angajat cu 4 sau 8 ore.

**Bibliografie**

1. **Suport de curs** (<http://acs.ase.ro/Media/Default/documents/designpatterns/Curs%20CTS%20%20Design%20Patterns%20v2.pdf> )
2. <http://acs.ase.ro/Media/Default/documents/cts/curs/Curs%20CTS%20%20-%20GIT.pdf>
3. <https://github.com/catalinboja/cts-sourcecode>
4. <http://git-scm.com/book/en/v2>