**ACADEMIA DE STUDII ECONOMICE DIN BUCUREŞTI  
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ**

**PROIECT   
CALITATE ȘI TESTARE SOFTWARE**

**BUSUIOC ROXANA  
GRUPA: 1049**

**2015**

**Cuprins**

1. Definirea si detalierea pattern-urilor utilizate
2. Definirea si detalierea metodelor testate prin Unit Testing
3. Definirea si descrierea Test Case-urilor
4. Definirea si descrierea Test Suite-ului
5. Descrierea sumara a functiilor aplicatiei cu referire la pattern-uri si la metode testate
6. **Descrierea pattern-urilor folosite**

Pattern-urile folosite in aplicatie sunt: Singleton, Factory si Observer.

Pattern-ul Singleton este utilizat pentru crearea unei singure instante (unica). Am folosit design pattern-ul creational Singleton pentru a crea o singura instanta in cazul conexiunii cu baza de date.

Pattern-ul Factory defineste o metoda pentru crearea de obiecte din aceeasi familie (interfata) in subclase. Este utilizat pentru a crea obiecte din familia Stoc: Recolta.class si ResursaExterna.class. Recolta reprezinta cantitatea recoltata, iar resursele externe sunt reprezentate de ceea ce trebuie achizitionat din exterior pentru buna desfasurare a culturii planificate.

Pattern-ul Observer defineste un handler pentru diferite evenimente. In aceasta aplicatie este utilizat pentru a notifica angajatii atunci cand este creata o activitate noua. Pentru implementarea pattern-ului am creat clasele Observer, Subject, Angajat si Activitate.

Interfata Observer contine metoda update(String statusActivitate). Aceasta functie este implementata in clasa Angajat care implementeaza interfata Observer.

Interfata Subject defineste metodele inregistreaza(Observer obs), sterge(Observer obs), notifica(). Aceste metode sunt implementate in clasa Activitate.

1. **Definirea si detalierea metodelor testate prin Unit Testing**
2. In clasa Achizitie am definit metoda publica float calculeazaValoareAchizitie(float pret, float cantitate). Metoda primeste ca parametrii pretul si cantitatea, de tip real si returneza produsul dintre acestia reprezentand valoarea achizitionata
3. In clasa Activitate am definit metoda publica void setStatus(String statusActivitate). In cadrul acestei metode am pus conditii ca tipul statusului unei activitati sa aiba doar valorile {“inceput”,”terminat”,”progress”}, altfel sa fie aruncata o exceptie prin care se afiseaza la consola mesajul “Status incorect!”.
4. In clasele Angajat, Client, Furnizor, Utilaj, Teren am definit metodele publice void insereazaAngajat(), insereazaClient(), insereazaFurnizor(), insereazaTeren(), insereazaUtilaj() care iau ca parametrii atributele obiectelor de tipul respectiv pentru a realiza inserarea in baza de date. In cadrul metodei instantiez un obiect de tip angajat cu ajutorul constructorului cu parametrii si apoi il inserez in baza de date.
5. In aceeleasi clase Angajat, Client, Furnizor, Utilaj, Teren am creat si metoda public ArrayList<Angajat> selecteazaAngajati(), ArrayList<Client> selecteazaClienti(),ArrayList<Furnizor> selecteazaFurnizori(),ArrayList<Utilaj> selecteazaUtilaje(), ArrayList<Teren> selecteazaTerenuri(), care iau ca parametru numele angajatului, clientului sau furnizorului cautat, respective localizarea terenului sau denumirea utilajului si selecteaza din baza de date toate inregistrarile care au acelasi nume ca si cel specificat. Metoda returneaza o lista de obiecte care coincid criteriului, iar in cazul in care nu exista nicio inregistrare care sa corespunda cerintelor cautarii se arunca o exceptie.
6. In clasa Client am definit metoda publica void setDenumireClient(String denumire) prin care verificam daca denumirea clientului a fost introdusa, in cazul in care nu a fost se aruca o exceptie.
7. In clasa Client am creat metoda public void setCUI(String CUI) prin care se verifica daca a fost introdus codul unic de identificare al clientului, in cazul in care nu a fost introdus este aruncata o exceptie: "Codul Unic de Identificare necompletat".
8. In clasa Client am definit metoda void setEmail(String email) prin care se verifica daca emailul este valid, in cazul in care nu este, va fi aruncata o exceptie. Pentru verificarea validitatii adresei de email am folosit o alta functie isValid(String email) care returneaza true daca emailul introdus corespunde pattern-ului definit, iar daca nu va returna false.
9. Tot in clasa Client am verificat daca numarul de telefon introdus contine doar cifre. Astfel am creat functia contineDoarCifre(String str) care returneaza true daca numarul de telefon este format doar din cifre si false daca nu. Am apelat aceasta functie in metoda void setNumarTelefon(String numarTelefon), in cazul in care numarul de telefon nu corespunde criteriului se arunca o exceptie: "Numar de telefon incorect!".
10. In clasa Comanda am definit functia publica double calculeazaValoareComanda(float pret, float cantitate,String tipClient), care returneaza valoarea comenzii facand produsul dintre pretul, cantitatea si adaosul commercial. In cazul in care clientul este de tip PF (persoana fizica) adaosul practicat va fi de 16%, iar in cazul in care clientul este de tip PJ (persoana juridical) adaosul practicat va fi de 24%.
11. In clasa Comanda am definit metoda double verificaValoareComanda(double valoareComanda), care aplica un discount in functie de valoarea comenzii. Asfel, daca valoarea comenzii este intre 500 si 1000 RON se va aplica un discount de 0.05, iar daca valoarea comenzi depaseste 1000 RON se aplica un discounde 0.1. In cazul in care valoarea comenzii este mai mica sau egala cu 0, se va arunca o exceptie. Functia returneaza valoarea discountului aplicat.
12. Tot in clasa Comanda am creat metoda void setTipClient care ia ca parametru tipul clientului si verifica daca este PF sau PJ (persoana fizica sau juridical). Daca acesta nu se incadreaza va fi aruncata o exceptie.
13. In clasa PlanificareCultura am definit functia double calculeazaRandament(double pretUnitar, double suprafata) care calculeaza produsul dintre pretul unitar al culturii si suprafata terenului. In cazul in care randamentul este negativ, va fi aruncata o exceptie. Functia returneaza valoarea randamentului calculat.
14. Clasa Recolta implementeaza interfata IStoc si implicit metodele din aceasta: float adauga(float cantitate) si float scade(float cantitateVanduta).

Metoda float adauga(float cantitate) actualizeaza stocul existent cu productia recoltata. In cazul in care cantitatea este mai mica decat 0, se va arunca o exceptie.

Metoda float scade(float cantitate) actualizeaza stocul existent in cazul vanzarilor. Daca stocul existent recoltata este mai mic decat 0 se va arunca o exceptie.

1. Clasa ResursaExterna implementeaza interfata IStoc si implicit metodele din aceasta: float adauga(float cantitate) si float scade(float cantitateVanduta).

Astfel float adauga(float cantitate) reactualizeaza resursele existente in stoc cu cantitatea achizitionata, iar float scade(float cantitate) scade din resursele existente ceea ce s-a consumat in urma desfasurarii activitatilor agricole.

1. In clasa Teren am implementat metoda float calculeazaCantitateSuprafataArabila() prin care suprafata totala este inmultita cu 0.9, rezultand astfel suprafata plantata/arabila.
2. In clasa Teren am definit metoda Boolean verificaCalculSuprafataArabila(float suprafataArabila) prin care se verifica rezultatul metodei calculeazaCantitateArabila(). Daca suprafata totala este mai mare decat suprafata arabila va returna true, iar daca nu va returna false.
3. **Definirea si descrierea Test Case-urilor**
4. TestAchizitie

* testValoareAchizitie() - verifica daca valoarea achizitie calculate prin functia calculeazaValoareAchizitie(float pret, float cantitate) este corecta.

1. TestActivitate

* testStatusIncorect() – verifica daca atunci cand se introduce statusul incorect, este aruncata exceptia.
* testStatusCorect() - verifica daca statusul introdus pentru activitate este correct.

1. TestCaseInsert

* testInserareAngajat() - verifica daca se insereaza un angajat nou in baza de date
* testInserareClient() - verifica daca se insereaza un client nou in baza de date
* testInserareFurnizor() - verifica daca se insereaza un furnizor nou in baza de date
* testInserareTeren() - verifica daca se insereaza un teren nou in baza de date

1. TestClient

* testNumarTelefonIncorect() - arunca o exceptia daca numarul de telefon contine si litere
* testNumarTelefonCorect() – verifica daca numarul de telefon introdus al clientului corespunde cu valoarea data.
* testEmailIncorect() - arunca o exceptie in cazul in care emailul nu corespunde formatului
* testEmailCorect() – verifica daca emailul introdus corespunde cu valoarea data.
* testCUICorect () – verifica daca CUI-ul introdus corespunde cu o valoare data.
* testCUIIncorect() – verifica exceptia in care codul de identificare nu este introdus
* testDenumireCorect()- verifica daca denumirea clientului corespunde cu cea introdusa
* testDenumireIncorect()- verifica cazul in care denumirea nu este completata si arunca o exceptie in aceasta situatie

1. TestComanda

* testCalculeazaValoareComanda() – verifica daca valoarea comenzii calcultate corespunde
* testVerificaProcentCorect() – calculeaza valoarea comenzii si verifica daca procentul aplicat este conform valorii comenzi.
* testVerificaValoareComandaZero() –verifica calcularea valorii comenzii in cazul in care pretul este negativ
* testVerificaValoareComandaMinus() – verifica calcularea valorii comenzii in care pretul este egal cu 0.
* testVerificaTipClientCorect() – verifica daca tipul clientului este PJ in cazul in care nu este arunca o exceptie.
* testVerificaTipClientIncorect() – arunca o exceptie daca tipul clientului nu are valoarea PJ sau PF.

1. TestFisier

* testCitireDinFisier() – verifica daca se executa citire din fisierul “terenuri.txt” si daca suprafata arabila citita din fisier are aceeasi valoare cu cea calculata prin intermediul functiei calculeazaSuprafataArabila().
* testFisierSuprafataArabila() - verifica daca valoarea rezultata in urma calculului suprafetei arabile este mai mic decat suprafata totala a terenului.
* testFisierSuprafataArabilaIncorect() - verifica daca valoarea rezulata in urma calculului suprafetei arabile este mai mic decat suprafata totala a terenului.

1. TestPlanificareCultura

* testCalculeazaRandament() – verifica daca valoarea randamentului calculat corespunde cu o valoarea data
* testCalculeazaRandamentIncorect() - arunca o exceptie in cazul in care randamentul este negativ.

1. TestStoc

* testAdaugaRecoltaCorect() - verifica daca valoarea recoltei adaugate corespunde cu o valoare data
* testAdaugaRecoltaIncorect() – arunca o exceptie in cazul in care valoarea recoltei este mai mica sau egala cu 0
* testCantitateVandutaCorect() - verifica daca s-a actualizat cantitatea din stoc in urma vanzarii
* testCantitateVandutaIncorect() - arunca o exceptie in cazul in care valoarea recoltei din stoc este mai mica sau egala cu 0
* testCantitateUtilizataCorect() - verifica daca s-a actualizat cantitatea din stoc in urma utlizarii
* testCantitateUtilizataIncorect() - arunca o exceptie in cazul in care cantitatea existent de resurse externe din stoc este mai mica sau egala cu 0
* testCantitateAchizitionataCorect() - verifica daca s-a actualizat cantitatea din stoc in urma achizitionarii
* testCantitateAchizitionataIncorect() - arunca o exceptie in cazul in care cantitatea achizitionata este mai mica sau egala cu 0

1. TestUtilajCRUD

* testInserareUtilaj() – verifica daca s-a efectuat inserarea in baza de date
* testGasesteUtilaj() - verifica daca in urma cautarii dupa denumire, denumirea primului utilaj din lista coincide cu valoarea data.
* testGasesteUtilajNuExista() - verifica daca se arunca o exceptie in cazul in care nu exista niciun utilaj in baza de date cu denumirea data.

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

Test Suite-ul AllTestsSuite contine toate Test Case-urile create in cadrul aplicatie. Am adaugat Test Case-uri atat partial, pentru TestFisier, TestPlanificareCultura si TestUtilajCRUD, cat si total pentru restul TestCase-urilor.

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

In vederea implementarii design pattern-ului Factory am utilizat clasele: interfata IStoc, Recolta, ResursaExterna si StocFactory. Am definit in interfata IStoc metodele float adauga(float cantitate) si float scade(float cantitate) pe care le-am implementat in clasele Recolta si ResursaExterna. In TestStoc am instantiat. un obiect de tip Recolta si unul de tip ResursaExterna.