ACADEMIA DE STUDII ECONOMICE

Facultatea: Cibernetică, Statistică şi Informatică Economică

Specializarea: Informatică economică

**PROIECT CALITATE ȘI TESTARE SOFTWARE**

STUDENT: VASILE LUCIAN

Bucureşti

2015

**CUPRINS**

1. Definirea și detalierea design pattern-urilor utilizate ................. 4
2. Definirea și detalierea metodelor testate prin Unit Testing ...... 5
3. Definirea și descrierea Test Case-urilor ............................................. 8
4. Definirea și descrierea Test Suite-ului .............................................. 10

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

**Singleton**

Primul design pattern folosit în proiect este Singleton. Acesta este caracterizat prin unicitatea sa, deoarece creează o singură instanță ce poate fi creată o singură dată, dar referită de mai multe ori. Singleton asigură un singur punct de acces, vizibil global, la unica instanță.

Exemplu:

**public static synchronized Masina getInstance() {**

**if (instance == null)**

**instance = new Masina("Skoda", "Octavia", "alb", 2010, 30000);**

**return instance;**

**}**

**Builder**

Al doilea design pattern utilizat este Builder. Acesta reprezintă o interfață abstractă ce definește metodele prin care sunt construite anumite părți ale obiectului complex. Un avantaj foarte mare al acestui pattern este acela că algoritmul de creare al obiectului este flexibil din toate punctele de vedere, iar clientul poate alege ce părți să fie create.

În proiect a fost apelată metoda construieste din interfața IMasinaBuiler în clasa MasinaBuilder, iar după setarea fiecărui parametru a acesteia se poate include în Clasa principală următoarea secvență de cod:

**MasinaBuilder constructorMasina = new MasinaBuilder();**

**constructorMasina.setMarca("Renault").setModel("Megane").setCuloare("albastru").setAnFabricatie(2005).setNrKm(150000);**

**Masina masina2 = constructorMasina.construieste();**

Rezultatul rulării: Masina [marca=Renault, model=Megane, culoare=albastru, anFabricatie=2005, nrKm=150000.0]

**Decorator**

Cel de-al treilea design pattern folosit este Decorator. Acesta gestionează o referință de tip abstract către obiectul decorat, iar metodele moștenite apelează implementările specifice din clasa obiectului referit. Decorarea se face pe mai multe niveluri și este transparentă pentru utilizator deoarece clasa moștenește interfața specifică obiectului. Acest pattern nu impune limite privind un număr maxim de decorări.

În proiect a fost utilizată metoda abstractă plecare, inițializată în clasa VanzareNoua și metoda vinde inițializată în interfața IMasina.

În clasa principală este inclusă următoarea secvență de cod:

**VanzareNoua vanzare = new Vanzare(new Masina("Opel", "Insignia", "rosu", 2013, 15000));**

**vanzare.getMasina().vinde();**

**vanzare.plecare();**

Rezultatul rulării:

Masina Opel Insignia fabricata in anul 2013 si rulata 15000.0 km a fost vanduta.

Masina Opel Insignia a parasit showroom-ul.

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

* Calcularea stocului de mașini după vânzarea unora dintre ele

Exemplu utilizat:

public int scadereNrMasini(int x, int y) {

return x - y;

}

* Calcularea stocului de mașini după primirea unui nou transport

Exemplu utilizat:

public int inmultireNrMasini(int x, int y) {

return x \* y;

}

* Calcularea rezultatului obținut prin ridicarea la pătrat a kilometrilor din bord a uneia dintre mașini

Exemplu utilizat:

public int patratNrKm(int x) {

return x \* x;

}

* Calcularea mediei aritmetice dintre anii de fabricație a mașinilor din showroom

Exemplu utilizat:

public int medieAritmeticaAniFabricatie(int x, int y) {

return (x + y) / 2;

}

* Calculare mediei geometrice dintre kilometrii ultimelor 2 mașini intrate în showroom

Exemplu utilizat:

public int medieGeometricaNrKm(int x, int y) {

return (int) Math.sqrt(x \* y);

}

* Identificarea mașinii cu numărul maxim de kilometrii dintre ultimele 2 vândute

Exemplu utilizat:

public int maximKm(int x, int y) {

return Math.max(x, y);

}

* Identificarea mașinii cu numărul minim de kilometrii dintre ultimele 2 vândute

Exemplu utilizat:

public int minimKm(int x, int y) {

return Math.min(x, y);

}

* Verificarea parității numărului de kilometrii din bordul uneia dintre mașini

Exemplu utilizat:

public boolean numarParKm(int x) {

if (x % 2 == 0)

return true;

else

return false;

}

* Verificarea imparității numărului de kilometrii din bordul uneia dintre mașini

Exemplu utilizat:

public boolean numarImparKm(int x) {

if (x % 2 == 1)

return true;

else

return false;

}

* Concatenarea unei mărci și a unui model de mașină

Exemplu utilizat:

public String concatenare(String unu, String doi) {

return unu + " " + doi;

}

* Numărarea apariției literei „a” într-o propoziție

Exemplu utilizat:

public int numara(String litera) {

int numara = 0;

for (int i = 0; i < litera.length(); i++) {

if (litera.charAt(i) == 'a' || litera.charAt(i) == 'A') {

numara++;

}

}

return numara;

}

* Calcularea sumei kilometrilor celor mai recent vândute mașini din showroom cu ajutorul jar-ului extern Mockito

Exemplu utilizat:

public class Calculator {

ICalculatorKmRulati calculatorBord;

public int adaugaDouaNumbere(int x, int y) {

return calculatorBord.adunaKm(x, y);

}

public ICalculatorKmRulati getCalc() {

return calculatorBord;

}

public void setCalc(ICalculatorKmRulati calc) {

this.calculatorBord = calc;

}

* Inmultirea si impartirea kilometrilor a 3 masini aflati intr-un vector cu o valoare introdusa de utilizator

public KmMasiniShowroom inmultire(double numar) {

return new KmMasiniShowroom(this.Km1 \* numar, this.Km2 \* numar, this.Km3 \* numar);

}

public KmMasiniShowroom impartire(double numar) {

return new KmMasiniShowroom(this.Km1 / numar, this.Km2 / numar, this.Km3 / numar);

}

* Calcularea vârstei unui cumpărător cu ajutorul Codului Numeric Personal

Exemplu utilizat:

public int calculeazaVarstaCumparator() {

return Calendar.getInstance().get(1) - (1900 + Integer.parseInt("" + CNPCumparator.charAt(1) + CNPCumparator.charAt(2)));

}

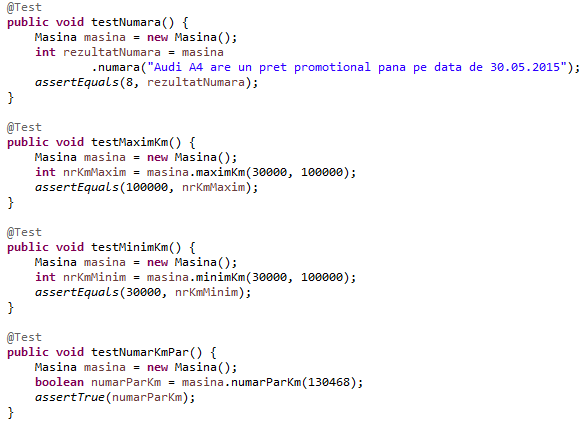
1. **Definirea și descrierea Test Case-urilor**

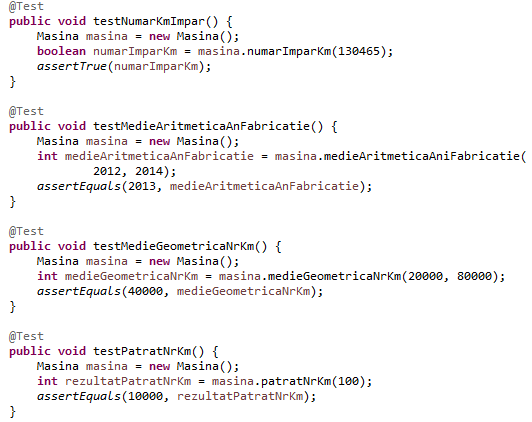
În proiect au fost folosite mai multe fișiere de test, fiecare fiind specific unei anumite clase.

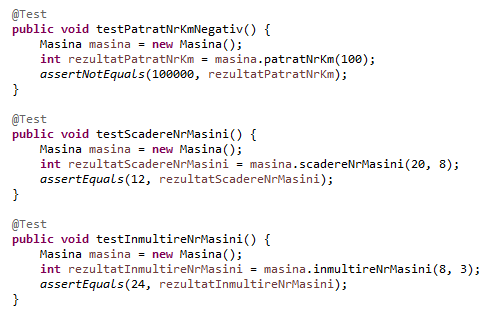
Un Test Case reprezintă o listă de probleme sau condiții setate de către utilizator în scopul obținerii diferitelor tipuri de testări asupra aplicației.

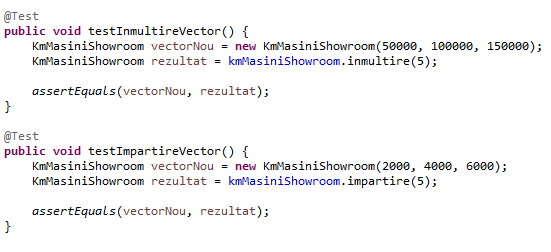
Exemple de teste utilizate:

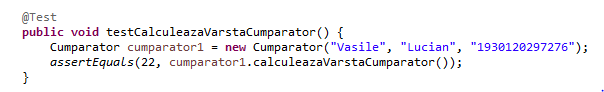


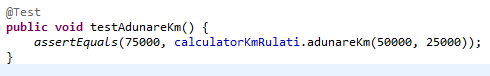








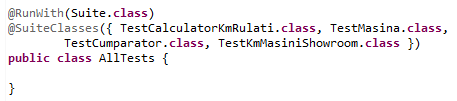




1. **Definirea și descrierea Test Suite-ului**

Un Test Suite reprezintă o colecție de teste. Acesta conține un număr de instrucțiuni detaliate pentru fiecare colecție de teste și metode prealabile sau descrieri ce țin de testele care urmează a fi efectuate în proiect.

Exemplu utilizat:



În imaginea de mai sus a fost creat un obiect de tip TestSuite care rulează absolut toate testele aflate în pachetul „teste”.