Ministerul Educaţiei al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Catedra Automatică şi Tehnologii Informaţionale

RAPORT

Lucrare de laborator nr 3

Disciplina: Ingineria produselor program

Tema: „Șabloane de proiectare de comportament”

A efectuat:

Vovc Artemie st. TI-133

A verificat:

Eugenia Latu lect. asist.

Chișinău 2016

Cuprins

1 Sarcina

De creat un program în care vor interacționa cinci șabloane de proiectare de comportament.

2 Șabloane de proiectare de comportament implementare

2.1 Observer

Exista componente care trebuie sa fie notificate la producerea unui eveniment

Gestiunea evenimentelor la nivel de interfață

Componentele se abonează/înregistrează la acel eveniment –modificare de stare/acțiune

La producerea unui eveniment pot fi notificate mai multe componente

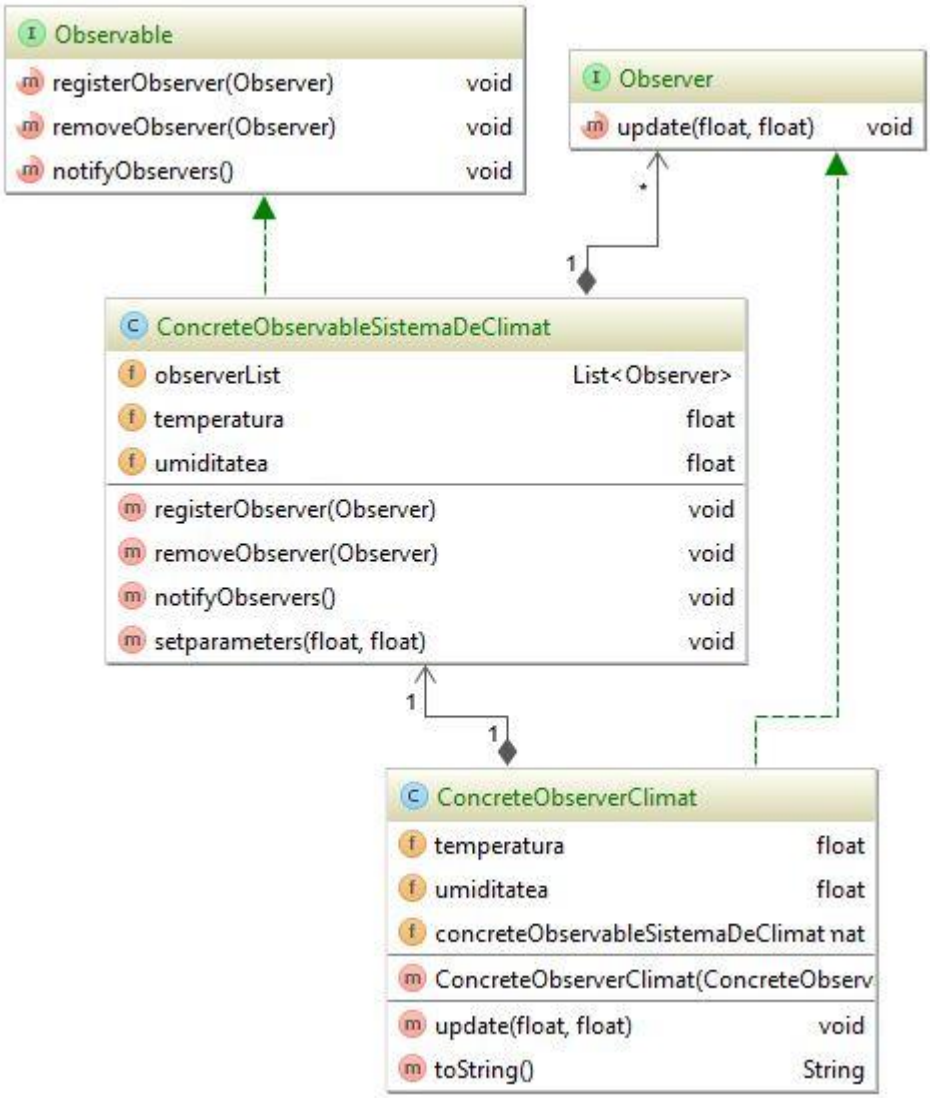

Figura 2.1 – observer

Externalizarea/delegarea funcțiilor către componente "observator" care dau soluții la anumite evenimente independent de proprietarul evenimentului

Conceptul integrat în pattern-ularhitectural *Model ViewController*(MVC)

Implementează conceptul POO de*loosecoupling*–obiectele sunt interconectate prin notificări și nu prin instanțieride clase și apeluri de metode

2.2 State

Aplicația tratează un anumit eveniment diferit în funcție de starea unui obiect

Numărul de stări posibile poate să crească și tratarea unitară a acestora poate să influențeze complexitatea soluției

Modul de tratare a acțiunii este asociat unei anumite stări și este încapsulat într-un obiect de stare

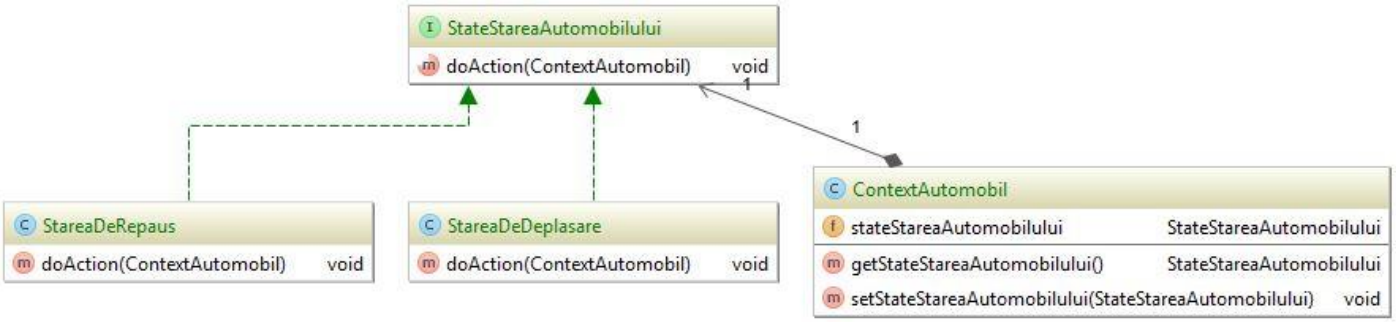

Figura 2.2 – state

2.3 Template

Implementarea unui algoritm presupune o secvență predefinita si fixa de pași

Metoda ce definește schema algoritmului –metoda template

Pot fi extinse/modificate metodele care implementează fiecare pas însă schema algoritmului nu este modificabilă

Implementează principiul Hollywood: "*Don'tcallus, we'llcallyou.*“

Metodele concrete de definesc pașii algoritmului sunt apelate de metoda template

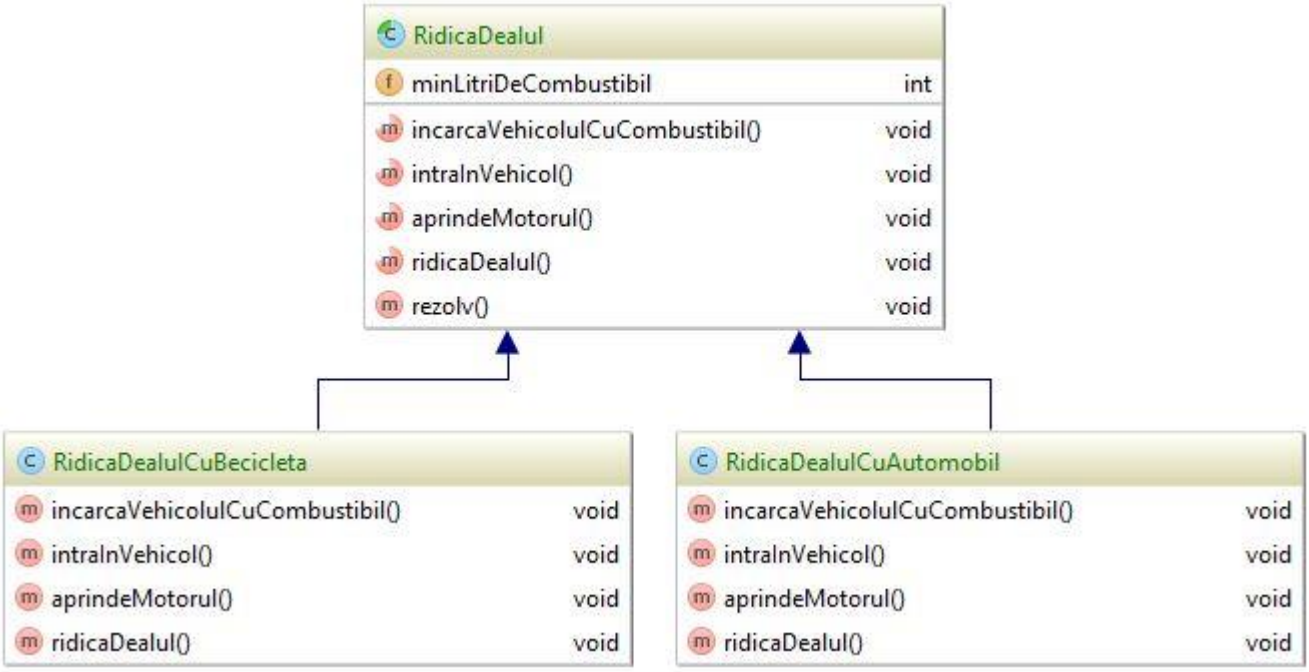

Figura 2.3 – template

2.4 Memento

Aplicația trebuie sa permită salvarea stării unui obiect

Imaginile stării obiectului pentru diferite momente sunt gestionate separat

Obiectul își poate restaura starea pe baza unei imagini anterioare

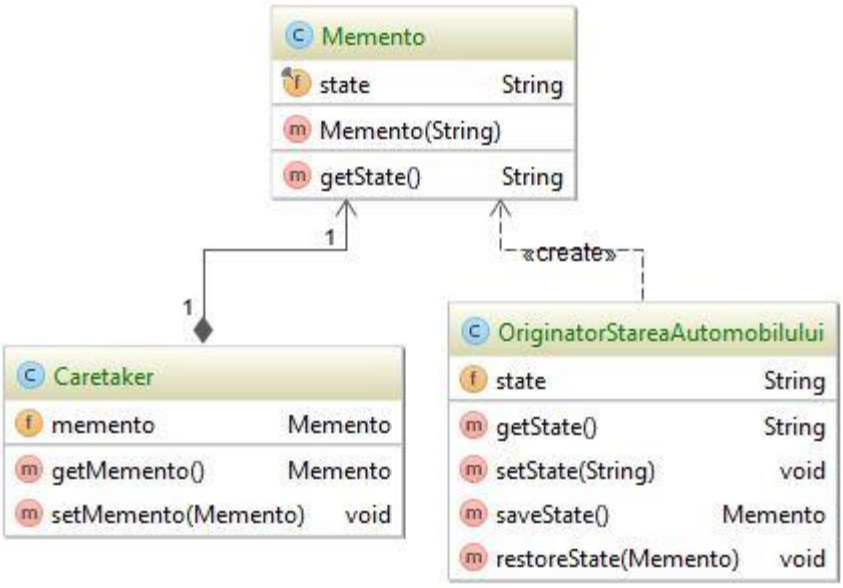

Figura 2.4 – memento

2.5 Iterator

In object-oriented programming, the iterator pattern is a design pattern in which an iterator is used to traverse a container and access the container's elements. The iterator pattern decouples algorithms from containers; in some cases, algorithms are necessarily container-specific and thus cannot be decoupled.

For example, the hypothetical algorithm *SearchForElement* can be implemented generally using a specified type of iterator rather than implementing it as a container-specific algorithm. This allows *SearchForElement* to be used on any container that supports the required type of iterator.

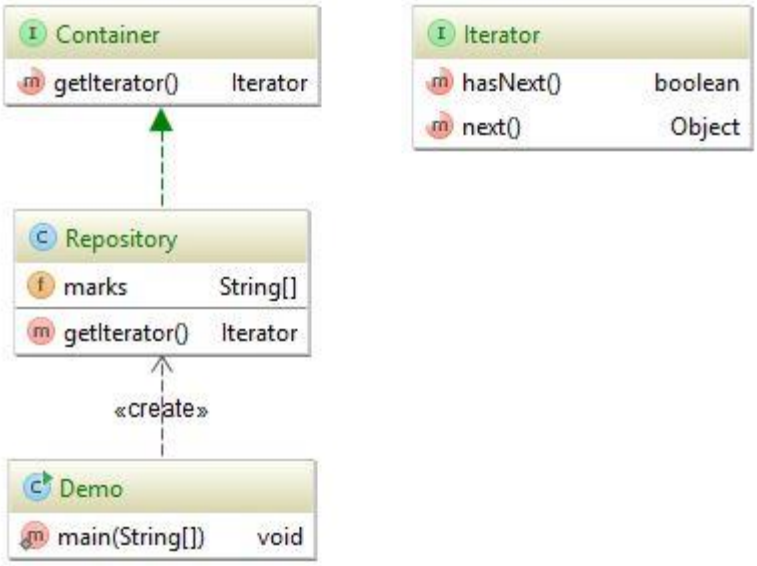

Figura 2.5 – proxy

Concluzia

Lucrarea dată a avut ca scop să ne facă cunoscuți cu șabloanele de proiectare de comportament, care sunt des utilizate în industria de dezvoltare a aplicațiilor. Am studiat cinci șabloane de proiectare de comportament: template, iterator, memento, state, observer. Fiecare tip de șablon are ajunsurile și neajunsurile. La general șabloanele studiate rezolvă unele probleme de comportament ale unui sistem informațional la nivel de implementare.

Bibliografia

1. Iterator. [Resursa electronică].-regim de acces: https://en.wikipedia.org/wiki/Iterator_pattern
2. State. [Resursa electronică].-regim de acces: https://en.wikipedia.org/wiki/State_pattern
3. Memento. [Resursa electronică].-regim de acces: https://en.wikipedia.org/wiki/Memento_pattern
4. Template. [Resursa electronică].-regim de acces: https://en.wikipedia.org/wiki/Template_method_pattern
5. Observer. [Resursa electronică].-regim de acces: https://en.wikipedia.org/wiki/Observer_pattern

Anexe A
Client

```
package com.lab3ipp;

import com.lab3ipp.adapter.AdapterAutospecialistSpecificat;
import com.lab3ipp.adapter.TargetAutospecialistul;
import com.lab3ipp.bridge.AbstractionAutomobil;
import com.lab3ipp.bridge.ConcreteImplementatorDacia;
import com.lab3ipp.bridge.RefinedAbstractionAutomobil;
import com.lab3ipp.composite.ComponentInterior;
import com.lab3ipp.composite.CompositeInterior;
import com.lab3ipp.composite.LeafScaun;
import com.lab3ipp.composite.LeafVolan;
import com.lab3ipp.facade.FacadeSuportTehnic;
import com.lab3ipp.proxy.ConduceAutomobilul;
import com.lab3ipp.proxy.ProxyConduce;

import java.util.Arrays;

/**
 * Created by Artemie on 24.09.2016.
 */
public class Client {
    public static void main(String[] args) {
        /**
         * ADAPTER
         */
        TargetAutospecialistul targetAutospecialistul = new
AdapterAutospecialistSpecificat();
        System.out.println("Motorul: " + targetAutospecialistul.schimbaMotorul() + "
Culoarea: " + targetAutospecialistul.schimbaCuloarea());
        /**
         * BRIDGE
         */
        AbstractionAutomobil abstractionAutomobil = new RefinedAbstractionAutomobil(new
ConcreteImplementatorDacia());
        System.out.println("Automobilul: " + abstractionAutomobil.getMarka());
        /**
         * COMPOSITE
         */
        ComponentInterior componentInterior = new
CompositeInterior().addComponent(Arrays.asList(new LeafVolan(), new LeafScaun()));
        System.out.println(componentInterior.get());
        /**
         * FACADE
         */
        FacadeSuportTehnic facadeSuportTehnic = new FacadeSuportTehnic();
        if(facadeSuportTehnic.trecut()) System.out.println("Automobilul a trecut
suportul tehnic.");
        /**
         * PROXY
         */
        ConduceAutomobilul conduceAutomobilul = new ProxyConduce().setIsEmpty(true);
        System.out.println("Conduc automobilul: " + conduceAutomobilul.get());

    }
}
```

Anexa B
Interfețele

```
package com.lab3ipp.iterator;

/**
 * Created by Artemie on 24.09.2016.
 */
public interface Container {
    Iterator getIterator();
}
package com.lab3ipp.iterator;

/**
 * Created by Artemie on 24.09.2016.
 */
public interface Iterator {
    boolean hasNext();

    Object next();
}
package com.lab3ipp.observer;

/**
 * Created by Artemie on 24.09.2016.
 */
public interface Observable {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();
}
package com.lab3ipp.observer;

/**
 * Created by Artemie on 24.09.2016.
 */
public interface Observer {
    void update (float temperature, float humidity);
}
package com.lab3ipp.state;

/**
 * Created by Artemie on 24.09.2016.
 */
public interface StateStareaAutomobilului {
    void doAction(ContextAutomobil contextAutomobil);
}
```

Anexa C
Clase

```
package com.lab3ipp.iterator;

/**
 * Created by Artemie on 24.09.2016.
 */
public class Repository implements Container {
    private String[] marks = {"toyota", "mercedes"};

    @Override
    public Iterator getIterator() {
        return new IteratorImpl();
    }

    private class IteratorImpl implements Iterator {
        int index;

        @Override
        public boolean hasNext() {
            return index < marks.length;
        }

        @Override
        public Object next() {
            if (this.hasNext()) return marks[index++];
            return null;
        }
    }
}
package com.lab3ipp.memento;

/**
 * Created by Artemie on 24.09.2016.
 */
public class Caretaker {
    private Memento memento;

    public Memento getMemento() {
        return memento;
    }

    public void setMemento(Memento memento) {
        this.memento = memento;
    }
}
package com.lab3ipp.memento;

/**
 * Created by Artemie on 24.09.2016.
 */
public class Memento {
    private final String state;

    public Memento(String state){
        this.state = state;
    }
```

```
    public String getState() {
        return this.state;
    }
}
package com.lab3ipp.memento;

/**
 * Created by Artemie on 24.09.2016.
 */
public class OriginatorStareaAutomobilului {
    private String state;

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public Memento saveState(){
        return new Memento(state);
    }

    public void restoreState(Memento memento){
        this.state = memento.getState();
    }
}
package com.lab3ipp.observer;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Artemie on 24.09.2016.
 */
public class ConcreteObservableSistemaDeClimat implements Observable {
    private List<Observer> observerList = new ArrayList<>();
    private float temperatura;
    private float umiditatea;

    @Override
    public void registerObserver(Observer o) {
        observerList.add(o);
    }

    @Override
    public void removeObserver(Observer o) {
        observerList.remove(o);
    }

    @Override
    public void notifyObservers() {
        for (Observer observer : this.observerList)
observer.update(temperatura,umiditatea);
    }

    public void setparameters(float temperatura,float umiditatea){
        this.temperatura = temperatura;
```

```
        this.umiditatea = umiditatea;
        notifyObservers();
    }
}
package com.lab3ipp.observer;

/**
 * Created by Artemie on 24.09.2016.
 */
public class ConcreteObserverClimat implements Observer {
    private float temperatura;
    private float umiditatea;
    private ConcreteObservableSistemaDeClimat concreteObservableSistemaDeClimat;

    public ConcreteObserverClimat(ConcreteObservableSistemaDeClimat
concreteObservableSistemaDeClimat) {
        this.concreteObservableSistemaDeClimat = concreteObservableSistemaDeClimat;
        concreteObservableSistemaDeClimat.registerObserver(this);
    }

    @Override
    public void update(float temperature, float umiditatea) {
        this.temperatura = temperature;
        this.umiditatea = umiditatea;
    }

    @Override
    public String toString() {
        return "ConcreteObserverClimat( temperatura= " + temperatura + ", umiditatea="
+ umiditatea + " );";
    }
}
package com.lab3ipp.state;

/**
 * Created by Artemie on 24.09.2016.
 */
public class ContextAutomobil {
    private StateStareaAutomobilului stateStareaAutomobilului;

    public StateStareaAutomobilului getStateStareaAutomobilului() {
        return stateStareaAutomobilului;
    }

    public void setStateStareaAutomobilului(StateStareaAutomobilului
stateStareaAutomobilului) {
        this.stateStareaAutomobilului = stateStareaAutomobilului;
    }
}
package com.lab3ipp.state;

/**
 * Created by Artemie on 24.09.2016.
 */
public class StareaDeDeplasare implements StateStareaAutomobilului {

    @Override
    public void doAction(ContextAutomobil contextAutomobil) {
        System.out.println("Automobilul se afla in starea de deplasare");
```

```
        contextAutomobil.setStateStareaAutomobilului(this);
    }
}
package com.lab3ipp.state;

/**
 * Created by Artemie on 24.09.2016.
 */
public class StareaDeRepaus implements StateStareaAutomobilului {
    @Override
    public void doAction(ContextAutomobil contextAutomobil) {
        System.out.println("Automobilul se afla in starea de repaus.");
        contextAutomobil.setStateStareaAutomobilului(this);
    }
}
package com.lab3ipp.template;

/**
 * Created by Artemie on 24.09.2016.
 */
public abstract class RidicaDealul {
    private int minLitriDeCombustibil = 0;
    public abstract void incarcaVehicolulCuCombustibil();
    public abstract void intraInVehicol();
    public abstract void aprindeMotorul();
    public abstract void ridicaDealul();

    public void rezolv(){
        this.incarcaVehicolulCuCombustibil();
        this.intraInVehicol();
        this.aprindeMotorul();
        this.ridicaDealul();
    }

}
package com.lab3ipp.template;

/**
 * Created by Artemie on 24.09.2016.
 */
public class RidicaDealulCuAutomobil extends RidicaDealul{
    @Override
    public void incarcaVehicolulCuCombustibil() {
        System.out.println("Deschid rezervorul");
        System.out.println("Umplu rezervorul");
        System.out.println("Inchid rezervorul");
    }

    @Override
    public void intraInVehicol() {
        System.out.println("Deschid usa");
        System.out.println("Intru in automobil");
        System.out.println("Inchid usa");
    }

    @Override
    public void aprindeMotorul() {
        System.out.println("Bag cheiele");
        System.out.println("Aprind motorul");
```

```
    }

    @Override
    public void ridicaDealul() {
        System.out.println("Apas pedala de accelerare");
        System.out.println("Ma deplases innainte");
    }
}
package com.lab3ipp.template;

/**
 * Created by Artemie on 24.09.2016.
 */
public class RidicaDealulCuBecicleta extends RidicaDealul {
    @Override
    public void incarcaVehicolulCuCombustibil() {
    }

    @Override
    public void intraInVehicol() {
        System.out.println("Ma asez pe bicicleta");
    }

    @Override
    public void aprindeMotorul() {

    }

    @Override
    public void ridicaDealul() {
        System.out.println("Invirt pedalele");
    }
}
```