LOCUINTA INTELIGENTA

Proiect Calitate si Testare Software

Iordache Razvan-Alexandru
Seria B Grupa 1069

Cuprins

Introducere

Acest proiect trateaza tema locuintei inteligente deoarece, conceptul de IoT (Internet of Things) este din ce in ce mai popular pe piata iar cererea de aplicatii care gestioneaza o locunta inteligenta sunt din ce in ce mai cautate.

Logica aplicatiei este urmatoarea:

Se doreste crearea unei singure instante de tip casa (locuinta) ce permite, printr-un mecanism centralizat ce creeaza obiecte de tip Incapare. Aceste incaperi pot fii de mai multe tipuri: incapere normala, bucatarie sau baie. Diferenta consta in metodele si atributele acestestora. Cu alte cuvinte baia spre deosebire de o camera normal este racordata la apa sau bucataria are alimentare cu gaze spre deosebire de baie.

Fiecare incapere ineligenta dispune de un numar mai mare de 3 atribute asa ca trebuie gasita o solutie pentru ca construi obiecte complexe printr-un mecanism ce permite clientului sa adauge valori fara a cunoaste continutul clasei si fara a incalca principiul de incapsulare.

Dupa cum am mentionat mai sus, unele incaperi dispun de alimentare cu apa, aceste incaperi folosesc un senzor Samsung ce contorizeaza fluxul de apa utilizat dar si de un mecanism de cost control dezvoltat de LG. Aplicatia trebuie sa permita utilizarea acestor senzori fara a modifica codul sursa a acestora.

Mecanismul de cost control trebuie sa fie limitat la ip-urile utilizatorilor conectati la reteaua wireless.

De asemeenea, fiecare incapere dispune de numeroase atribute astfel trebuie gasita o solutie pentru a primi usor informatii generale despre casa la un moment dat.

Casa inteligenta mai dispune si de un sistem anti-incendiu ce monitorizeaza temperature din fiecare camera si notifica utilizatorul in cazul in care temperatura trece de un anumit prag. Notificarea se face pe niveluri la fel ca si logarea de mesaje (trace, info, warning, error) in functie de gravitatea situatiei.

Casa inteligenta poate dispune si de un garaj. Situatia in cazul garajului este diferita de cea dintr-o incapere inteligenta. Garajul se poate afla in diferite stari. De exemplu daca numarul de inmatriculare corespunde atunci masina este acceptata, daca se afla o masina in garaj poate incepe un program de spalare daca utilizatorul doreste.

1. Singleton:

Acest pattern este utilizat pentru crearea unei singure instante de tip casa in aplicaie. Implementarea acestui pattern necesita ascunderea costructorului si asigurarea unui punct de acces vizibil global ce permite crearea unei singure instante.

47 lines (35 sloc) | 903 Bytes — Raw | Blame | History

```
1  package application;
2
3  public class Casa {
4
5          final String denumire;
6          private static int nrIncaperi=0;
7
8          private static Casa singleton=null;
9
10         private Casa(String denumire) {
11                 super();
12                 this.denumire = denumire;
13         }
14
15         public static Casa getCasa(String Denumire)
16         {
17                 if(singleton == null && Denumire != null)
18                 {
19                         singleton=new Casa(Denumire);
20                 }
21                 else if( Denumire == null)
22                 {
23                         throw new NullPointerException("Denumirea nu poate fii nula");
24                 }
25
26                 return singleton;
27
28         }
```

Fig 1 – Implementare Singleton

2. Factory

Cu ajutorul pattern-ului Factory implementam un mecanism centralizat prin care se construiesc obiecte. Obiectele sunt de tip incapere inteligenta si ele pot fii: camera normala, bucatarie sau baie.

Aceasta solutie este scalabila in sensul ca, se pot adauga noi obiecte concrete fara a afecta cu nimic codul sursa existent.

```
public IncapereInteligenta CreateIncapere(TipIncapere tip, String Nume, String mp) throws Exception
{
        switch(tip)
        {
        case BAIE:
                return new Baie(Nume, mp, 20, false, true, false, true, false);
        case BUCATARIE:
                return new Bucatarie(Nume,mp,24,true,true,false,true,false);
        case CAMERANORMALA:
                return new CameraNormala(Nume,mp,24,true,true,false,true);
                default: throw new Exception();
        }
}
```

Fig 2 – Factory

3. Builder

Deoarece o incapere inteligeta are mai mult de 2-3 atribute este dificil de construit un obiect. Asadar patter-ul Builder ne ajuta sa construim un obiect simplu, cu un minim de atribute iar ulterior sa apelam niste metode cu nume sugestiv pentru a adauga alte atribute obiectului definit.

43 lines (30 sloc) | 725 Bytes — Raw | Blame | History

```
1  package application;
2
3  public class BuilderCamera {
4
5          private CameraNormala camera=null;
6
7          public BuilderCamera(String nume,String suprafata) throws Exception {
8                  super();
9                  if(nume != null && suprafata != null)
10                         this.camera = new CameraNormala(nume, suprafata, 25, false,false,false, false);
11                 else
12                         throw new Exception("Nu se accepta null");
13         }
14
15         public CameraNormala build()
16         {
17                 return this.camera;
18         }
19
20         public BuilderCamera AreGaze()
21         {
22                 this.camera.setSursaGaze(true);
23                 return this;
24         }
25
26         public BuilderCamera AreSursaDeApa()
27         {
28                 this.camera.setSursaApa(true);
29                 return this;
30         }
31
32         public BuilderCamera AprindeLumina() throws ExceptieLipsaCurent
33         {
34                 this.camera.DeschideLumina();
35                 return this;
36         }
37
38
39
40
41
42 }
```

Fig 3 – Implementare Builder

Un exemplu de apel:

```
//Test Builder
try {
        System.out.println("Test Builder");
        CameraNormala cameraOaspeti= new BuilderCamera("Camera Oaspeti", "16.9 ").AprindeLumina().build();
        cameraOaspeti.PornesteAC(32);
        System.out.println(cameraOaspeti.getInfo());

        Camere.add(cameraOaspeti);
} catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4. Adapter

Unele incaperi dispun de alimentare cu apa, aceste incaperi folosesc un senzor Samsung ce contorizeaza fluxul de apa utilizat dar si de un mecanism de cost control dezvoltat de LG. Aplicatia trebuie sa permita utilizarea acestor senzori fara a modifica codul sursa a acestora, astfel intervine patter-ul Adapter ce construieste o punte intre metodele din interfata A si metodele din interfata B.

```
1  package application;
2
3  import java.util.stream.IntStream;
4
5  import LG.InterfataTrakerCostApa;
6  import samsung.InterfataSamsung;
7
8  public class AdaptorSamsungToLG implements InterfataTrakerCostApa{
9
10         InterfataSamsung interfataSamsung;
11
12         public boolean Proxy(String ip)
13         {
14                 if(ip.substring(0,3).equals("192"))
15                         return true;
16                 else
17                         return false;
18         }
19
20
21         public AdaptorSamsungToLG(InterfataSamsung interfataSamsung) {
22                 this.interfataSamsung = interfataSamsung;
23         }
24
25
26         @Override
27         public double getCost(String ip, double PretperL) throws Exception {
28
29                 if(PretperL<=0)
30                         throw new Exception("Pret per L trebuie sa fie > 0");
31                 if(Proxy(ip))
32                 {
33                 int[] consum =new int[7];
34                 consum = interfataSamsung.getConsumApaSaptamanal(ip);
35                 int suma=IntStream.of(consum).sum();
36                 return suma*PretperL;
37                 }
38                 else throw new Exception("Invalid IP");
39
40         }
41
```

Fig 5 – Mod de implementare pattern Adapter

5. Proxy

Patter-ul Proxy controleaza accesul la metoda de cost control prin limitarea ip-urilor dupa cum se poate observa in Fig 5.

```
//Test Adaptor & Proxy
InterfataSamsung SenzorApa=new SenzorApaSamsung();
AdaptorSamsungToLG adaptor= new AdaptorSamsungToLG(SenzorApa);

try {
        System.out.println("Test Proxy & Adaptor");
        System.out.println("Cost total Apa: "+adaptor.getCost("192.168.0.2",4.3));
} catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

Fig 6 – Apel Adapter & Proxy

6. Façade

De asemeenea, fiecare incapere dispune de numeroase atribute astfel trebuie gasita o solutie pentru a primi usor informatii generale despre casa la un moment dat. Façade Definește o interfața simplificata pentru contextul existent.

```
public class FacadeCasaInteligenta {

        public static String status(List<IncapereInteligenta> Camere){
                StringBuffer statusCasa =
                                new StringBuffer();
                statusCasa.append("Start Facade ...");
                statusCasa.append("\n");
                statusCasa.append("Status la ora:"
                                +(new Date()).toLocaleString());
                statusCasa.append("\n");

                for( int x=0 ; x<Camere.size(); x++)
                {
                        statusCasa.append(Camere.get(x).getInfo());
                        statusCasa.append("\n");
                }

                InterfataSamsung SenzorApa=new SenzorApaSamsung();
                AdaptorSamsungToLG adaptor= new AdaptorSamsungToLG(SenzorApa);

                try {
                        statusCasa.append("Cost total Apa: "+adaptor.getCost("192.168.0.2",4.3));
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return statusCasa.toString();

        }
```

Fig 7 – Implementare Façade

7. Chain of Responsibility

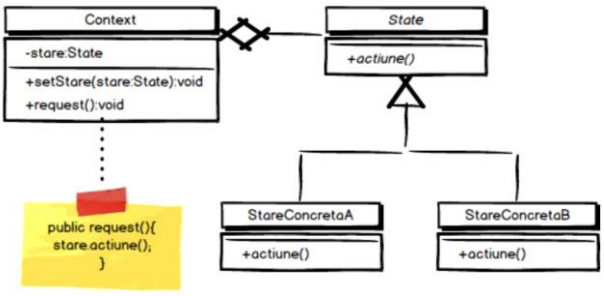
Casa inteligenta mai dispune si de un sistem anti-incendiu ce monitorizeaza temperature din fiecare camera si notifica utilizatorul in cazul in care temperatura trece de un anumit prag. Notificarea se face pe niveluri la fel ca si logarea de mesaje (trace, info, warning, error) in functie de gravitatea situatiei. In contextual existent Chain of Responsibility foloseste o clasa abstracta Handler pentru tratarea evenimentelor ce este implementata concret in AlarmaTemperatura

```
package Alarm;

public class AlertaTemperatura extends HandlerUrgente{

	@Override
	public void alertaTemperatura(StareTemperatura temperaturaCamera) {
		if(temperaturaCamera.temperatura <= 35)
		{
			if(this.next!=null)
				this.next.alertaTemperatura(temperaturaCamera);
		}
		else
			if(temperaturaCamera.temperatura <=45)
			{
				System.out.println("Camera necesita ventilatie");
				if(this.next!=null)
					this.next.alertaTemperatura(temperaturaCamera);
			}
			else
				System.out.println("Boss arde ceva !!!");

	}

}
```

Fig 8 – Chain of Responsibility : Alarma Temeratura

8. State

Casa inteligenta poate dispune si de un garaj. Situatia in cazul garajului este diferita de cea dintr-o incapere inteligenta. Garajul se poate afla in diferite stari. De exemplu daca numarul de inmatriculare corespunde atunci masina este acceptata, daca se afla o masina in garaj poate incepe un program de spalare daca utilizatorul doreste.



In cazul nostru Contextul este Garage si are urmatoarea implementare:

```
public class Garage {

	SmartGarageState isOpen;
	SmartGarageState canStore;
	SmartGarageState giveCar;
	SmartGarageState wash;

	SmartGarageState garageState;

	boolean doorOpen=false;

	public Garage()
	{
		isOpen= new IsOpen(this);
		canStore=new CanStore(this);
		giveCar=new ReturnCar(this);
		wash=new StartWashProgram(this);

		garageState=isOpen;
	}

	void setGarageState(SmartGarageState newGarageState)
	{
		garageState=newGarageState;
	}

	public void OpenGarageDoor()
	{
		garageState.openGarageDoor();
	}

	public void EjectCar(String NrInmatriculare)
	{
		garageState.ejectCar(NrInmatriculare);
	}

	public void StoreCar(String NrInmatriculare)
	{
		garageState.storeCar(NrInmatriculare);
	}

	public void StartWash()
	{
		garageState.cleanCars();
	}

	public SmartGarageState getYesOpenDoor() { return isOpen; }
	public SmartGarageState getYesCanStore() { return canStore; }
	public SmartGarageState getCarBack() { return giveCar; }
	public SmartGarageState getAWash() { return wash; }
```

Fig 9 – Context from State diagram

SmartGarageState este clasa abstracta ce definește interfața obiectelor ce gestionează implementarea unor acțiuni în funcție de stare.

```
public interface SmartGarageState {

        void openGarageDoor();
        void storeCar(String NrInmatriculare);
        void ejectCar(String NrInmatriculare);
        void cleanCars();
```

Fig 10 – State interface from State Diagram

JUnit Testing:

Fiind dat codul aplicatiei, sa se implmenenteze un sistem de testare automata, bazat pe JUnit, care sa verifice urmatoarele conditii:

TestCase 1:

1. Sa se verifice daca implementarea de Singleton este corecta si nu accepta argument null.

```
14  public class TestFactoryObjects extends TestCase {
15
16          Casa a;
17          Casa b;
18          CameraNormala Dormitor;
19          CameraNormala CameraCopil;
20          Baie Toaleta;
21          Bucatarie Bucatarie;
22          int [] numbers;
23
24          protected void setUp() throws Exception {
25                  super.setUp();
26                  a=Casa.getCasa("Razvan");
27                  b=Casa.getCasa("Razvan");
28
29                  Dormitor=(CameraNormala)a.CreateIncapere(TipIncapere.CAMERANORMALA, "Dormitor", "24.6");
30                  CameraCopil=(CameraNormala)a.CreateIncapere(TipIncapere.CAMERANORMALA, "Camera Copil", "16.4");
31                  Toaleta=(Baie)a.CreateIncapere(TipIncapere.BAIE,"Toaleta Principala", "12.5");
32                  Bucatarie=(Bucatarie)a.CreateIncapere(TipIncapere.BUCATARIE, "Bucatarie ", "24");
33
34                  Scanner scanner = new Scanner(new File("C:\\Users\\Razvan\\workspace\\ProiectCTS\\src\\Testing\\FactoryData.txt"));
35                  numbers= new int [100];
36                  int i = 0;
37                  while(scanner.hasNextInt()){
38                     numbers[i++] = scanner.nextInt();
39                  }
40
41          }
42
43          public void testSingleton()
44          {
45
46                  assertEquals(a.hashCode(),b.hashCode());
47
48          }
```

2. Folosind un fisier de resurse sa se testeze seter-ul pentru temperature sa nu primeasca valori aberante.

```
64      public void testSetariObiecte()
65      {
66              Dormitor.setUsaDeschisa(true);
67              assertEquals(true, Dormitor.isUsaDeschisa());
68
69              //Testam setari temperatura
70              for( int i=0; i<numbers.length; i++ )
71              {
72                      //Temperatura acceptata intre 5-60 Grade
73                      Dormitor.setTemperatura(numbers[i]);
74                      assertEquals(numbers[i], Dormitor.getTemperatura());
75
76                      Toaleta.setTemperatura(numbers[i]);
77                      assertEquals(numbers[i], Toaleta.getTemperatura());
78
79                      Bucatarie.setTemperatura(numbers[i]);
80                      assertEquals(numbers[i], Bucatarie.getTemperatura());
81              }
82
83              Bucatarie.setSursaApa(false);
84              assertEquals(false,Bucatarie.isSursaApa());
85
86
87      }
```

3. Folosind acelasi fisier de date sa se testeze metoda PornesteAC.

```
88
89      public void testMedodeIncapereInteligenta()
90      {
91              for( int i=0; i<numbers.length; i++ )
92              {
93                      try {
94
95                              Dormitor.PornesteAC(numbers[i]);
96                              System.out.println("Acceptam valoarea "+numbers[i]+" pentru Porneste AC ");
97                      } catch (Exception e) {
98                              // TODO Auto-generated catch block
99                              //System.out.println("Nu este acceptata valoarea "+numbers[i]);
100
101                     }
102
103             }
104     }
```

TestCase 2:

1. Sa se testeze daca clasa builder trateaza cazul in care argumentele sunt nulle.

```
8   public class TestBuilder extends TestCase {
9
10          CameraNormala cameraOaspeti;
11          protected void setUp() throws Exception {
12                  super.setUp();
13
14          }
15
16          public void testBuilder()
17          {
18                  String gol=null;
19                  try {
20                          cameraOaspeti= new BuilderCamera(gol, gol).AprindeLumina().build();
21                          fail();
22                  } catch (Exception e) {
23                          System.out.println("Builder trateaza null");
24                  }
25          }
26
27          protected void tearDown() throws Exception {
28                  super.tearDown();
29          }
30
31  }
```

Sa se creeze un TestSuit ce inglobeaza testele precedente pentru pattern-urile creationale.

17 lines (12 sloc) | 387 Bytes — Raw | Blame | History

```
1   package Testing;
2
3   import junit.framework.Test;
4   import junit.framework.TestSuite;
5
6   public class CreationalTestSuites extends TestSuite {
7
8           //Compose test suites and refer to them by class.
9           public static Test suite() {
10                  final TestSuite s = new TestSuite();
11                  s.addTestSuite(TestFactoryObjects.class);
12                  s.addTestSuite(TestBuilder.class);
13                  return s;
14          }
15  }
16
```

TestCase 3:

1. Testati metoda de cost control din AdapterSamsungToLG folosind multiple valori fie ele si aberante. (Ex. Pretul per unitate negative)

```
10          InterfataSamsung SenzorApa;
11          AdaptorSamsungToLG adaptor;
12
13          protected void setUp() throws Exception {
14                  super.setUp();
15                  SenzorApa=new SenzorApaSamsung();
16                  adaptor= new AdaptorSamsungToLG(SenzorApa);
17
18          }
19
20          public void testAdaptor()
21          {
22
23                  double[] valori=new double[] {1002 , 94 , 0 , 431 , -342.4 , -0.1 , 9 , 1000.55 , 0.02 };
24
25
26                          for (int i= 0 ; i<valori.length; i++ )
27                          {
28                                  try {
29                                          System.out.println("Pret: "+ valori[i]);
30                                          System.out.println("Cost: "+adaptor.getCost("192.168.0.2",valori[i]));
31                                          if(adaptor.getCost("192.168.0.2",valori[i])<= 0)
32                                                  fail();
33                                          }
34                                  catch (Exception e) {
35                                          // TODO Auto-generated catch block
36                                          System.out.println("Succes");
37                                  }
38                          }
39          }
```

2. Testati Proxy-ul din toate punctele de vedere pentru asigurarea unei securitati bune.

```
40
41          public void testProxy()
42          {
43                  String ip=null;
44                  try {
45                          adaptor.getCost(ip,5);
46                          fail();
47                  } catch (Exception e) {
48                          // TODO Auto-generated catch block
49                          System.out.println("Proxy denid null ip");
50                  }
51
52          }
```

TestCase 4:

1. Sa se testeze sistemul Anti-Incendiu (Chain of Responsibility) folosind un fisier de resurse. Observati daca gestionarea evenimentelor este corecta in functie de temperature setata in camera.

```
public class TestSistemAntiIncendiu extends TestCase {

	SistemDePrevenireAIncendiilor sistemAntiIncendiu;
	CameraNormala Dormitor;
	int [] numbers;
	protected void setUp() throws Exception {
		super.setUp();

		Casa a=Casa.getCasa("Razvan");
		sistemAntiIncendiu= new SistemDePrevenireAIncendiilor(new AlertaTemperatura());
		Dormitor=(CameraNormala)a.CreateIncapere(TipIncapere.CAMERANORMALA, "Dormitor", "24.6");

	}

	public void testChain()
	{
		try {
			Dormitor.PornesteAC(21);
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		Dormitor.setTemperatura(0);
		System.out.println(Dormitor.getTemperatura());

		Scanner scanner;
		try {
			scanner = new Scanner(new File("C:\\Users\\Razvan\\workspace\\ProiectCTS\\src\\Testing\\FactoryData.txt"));
			numbers= new int [100];
			int i = 0;
			while(scanner.hasNextInt()){
			   numbers[i++] = scanner.nextInt();
			}
		}
			catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}


		for( int x=0; x<numbers.length; x++ )
		{
			Dormitor.setTemperatura(numbers[x]);

		if(Dormitor.getTemperatura()>60 && Dormitor.getTemperatura()<5)
			fail();
		}
		System.out.println(Dormitor.getTemperatura());
		StareTemperatura newStare= new StareTemperatura(Dormitor.getTemperatura());
		sistemAntiIncendiu.procesare(newStare);

	}

	protected void tearDown() throws Exception {
		super.tearDown();
	}

}
```

Resurse proiect: https://github.com/razvaniordache/ProiectCTS