Factory Method Patroia

Eskatzen da: Aplikazioa aldatu negozio logikako objektuaren lorpena faktoria objektu batean zentralizatuta egoteko, eta aurkezpenak zein negozio logikako inplementazio erabili erabaki dezatela. Diseina eta inplementatu ebazpena Creator, Product eta ConcreteProduct jokatzen duten klaseen rolak garbi aurkeztuz.

BusinessLogicpaketeanFactory.javaklaseasortukodugu,bertancreateFactory() metodoa izango dugu.

*BLFactory.java × ApplicationLauncher.java

```
1  package businessLogic;
2
3  import java.net.MalformedURLException;
14
15 public class BLFactory {
16
17     public static BLFacade getBusinessLogicFactory(boolean isLocal) {
18         if (isLocal) {
19             DataAccess da = new DataAccess();
20             return new BLFacadeImplementation(da);
21         } else {
22             try {
23                 ConfigXML c = ConfigXML.getInstance();
24                 String serviceName = "http://" + c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort() + "/ws/"
25                         + c.getBusinessLogicName() + "?wsdl";
26                 URL url = new URL(serviceName);
27                 // 1st argument refers to wsdl document above
28                 // 2nd argument is service name, refer to wsdl document above
29                 QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
30
31                 Service service = Service.create(url, qname);
32
33                 return service.getPort(BLFacade.class);
34             } catch (Exception e) {
35                 e.printStackTrace();
36                 return null;
37             }
38         }
39
40     }
41
42 }
```

ApplicationLauncher.java -ren baldintza Factory.java klasean egingo dugunez, aldatu beharko dugu Factory klasera deitzeko.

*BLFactory.java ApplicationLauncher.java

```java
package gui;

import java.net.URL;

public class ApplicationLauncher {
    private static final Logger logger = Logger.getLogger(ApplicationLauncher.class.getName());

    public static void main(String[] args) {

        ConfigXML c = ConfigXML.getInstance();

        logger.info("Locale: " + c.getLocale());

        Locale.setDefault(new Locale(c.getLocale()));

        System.out.println("Locale: " + Locale.getDefault());

        try {

            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

            BLFacade appFacadeInterface = new BLFactory().getBusinessLogicFactory(c.isBusinessLogicLocal());

            MainGUI.setBussinessLogic(appFacadeInterface);
            MainGUI a = new MainGUI();
            a.setVisible(true);

        } catch (Exception e) {
            // a.jLabelSelectOption.setText("Error: "+e.toString());
            // a.jLabelSelectOption.setForeground(Color.RED);

            System.out.println("Error in ApplicationLauncher: " + e.toString());
        }

    }

}
```

ApplicationLauncher kodean, BLFactory erabiltzen da BLFacade interfaze baten instantzia lortzeko; BLFacadek negozio-logika ordezkatzen du. BLFactory-k erabakitzen du BLFacaderen tokiko edo urruneko inplementazioa itzuli, c.isBusinessLogicLocal() balio boolearraren arabera. Horrela, ApplicationLauncher-ek ez du zertan inplementazioaren xehetasunekin arduratu (tokikoa edo urrunekoa) eta BLFacade interfazearekin bakarrik lan egiten du. Honek kodea malguagoa eta askatuagoa izatea ahalbidetzen du.

Iterator patroia

Eskatzen da: Iteratzaile Hedatua inplementatu, eta adibidezko antzeko programa bat inplementatuz, hiriak aurkeztutako ordenan inprimatu.

Iterator patroia jarraituz, ExtendedIterator interfazea sortu dugu BLFacadeImplementation klasea hedatzeko.

```
package businessLogic;
import java.util.Iterator;
public interface ExtendedIterator<String> extends Iterator<String> {
		//return the actual element and go to the previous
		public Object previous();
		//true if ther is a previous element
		public boolean hasPrevious();
		//It is placed in the first element
		public void goFirst();
		//It is placed in the last element
		public void goLast();
}
```

Bertan, Iterator klasearen oinarrizko funtzioak definituta daude eta hurrengo ExtendedIteratorCities klasean metodo hauen inplementazioa agertzen da. Ikus daitekeenez, ExtendedIteratorCities klasearen eraikitzailean bi atributu hasieratu ditugu, departingCities izeneko zerrenda bat eta zerrendaren hasierako posizioa. Atributu hauen laguntzaz baliatuko gara aurreko interfazean aipatutako metodoak aurrera eramateko.

```
package businessLogic;
import java.util.List;
public class ExtendedIteratorCities implements ExtendedIterator<String> {
  private int pos;
  private List<String> departingCities;
  public ExtendedIteratorCities(List<String> departingCities) {
    this.pos = 0;
    this.departingCities = departingCities;
  }
  @Override
  public boolean hasNext() {
    return this.pos < this.departingCities.size();
  }
  @Override
  public String next() {
    if (hasNext()) {
        return this.departingCities.get(this.pos++);
    }
    return null;
  }
  @Override
  public String previous() {
    if (hasPrevious()) {
        return this.departingCities.get(--this.pos);
    }
    return null;
  }
```

```
  @Override
  public boolean hasPrevious() {
     return this.pos > 0;
  }
  @Override
  public void goFirst() {
     this.pos = 0;
  }
  @Override
  public void goLast() {
     this.pos = this.departingCities.size();
  }
}
```

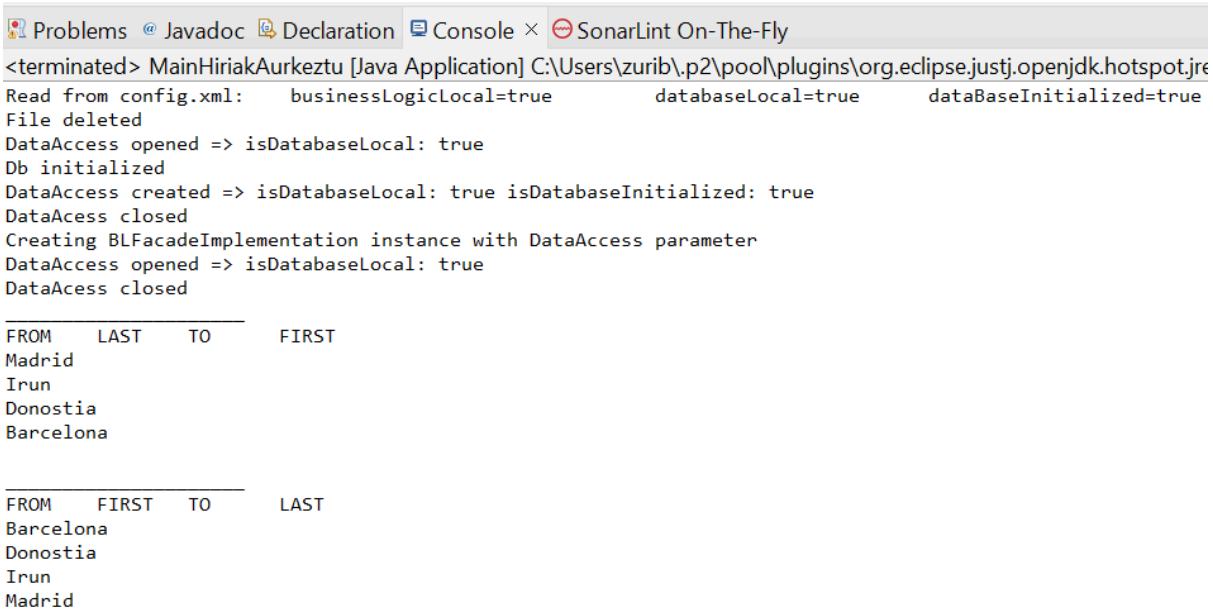
Behin metodo horiek inplementauta, BLFacadeImplementation klasean hurrengo metodoa gehitu dugu (getDepartingCitiesIterator) public List<String> getDepartingCities(); metodoa hedatzeko. Metodo honek inplementatuako klaseari deitzen dio eta interfazea bueltatzen du.

```
@WebMethod
public ExtendedIterator<String> getDepartingCitiesIterator(){
        return new ExtendedIteratorCities(getDepartCities());
}
```

Azkenik, main programa bat inplementatu dugu, MainHiriakAurkeztu izenekoa, DepartingCities eskatutako ordenean inprimatzen direla egiaztatzeko. Lehendabizi DepartingCities alderantzizko ordenan korritzen dira, eta jarraian ohiko ordenan.
Metodo honetan, Factory Method patroian garatutako BLFactory klaseari deitu diogu BLFacade lokalean exekutatzeko eta bertan definitutako getDepartingCitiesIterator() metodoari deitu diogu nahi ditugun Hirien zerrendak lortzeko.

```
package businessLogic;
public class MainHiriakAurkeztu {
        public static void main(String[] args) {
                //the BL is local
                boolean isLocal = true;
                BLFacade  blFacade =new BLFactory().getBusinessLogicFactory(isLocal);
                ExtendedIterator<String> i = blFacade.getDepartingCitiesIterator();
                String c;
                System.out.println("_____");
                System.out.println("FROM      LAST      TO      FIRST");
                i.goLast(); // Go to last element
                while (i.hasPrevious()) {
                        c = (String) i.previous();
                        System.out.println(c);
                }
                System.out.println();
                System.out.println("_____");
                System.out.println("FROM      FIRST      TO      LAST");
                i.goFirst(); //Go to first element
                while (i.hasNext()) {
                        c = i.next();
                        System.out.println(c);
                }
        }
}
```

Aurreko programaren exekuzioaren emaitza hurrengoa da:



Adapter patroia

Eskatzen da: DriverTable klasean deitzen den DriverAdapter klasea inplementatu, hasieran agertzen den taula aurkezteko (UML diseinua ere aurkeztu beharko da Adapter patroian parte hartzen duten klase guztiak aurkeztuz).

Hasteko, patroi honentzako BusinessLogic paketean egingo dugu eta bertan DriverAdapter.java sortuko dugu, non AbstractTableModel hedatuko duen. Klase honetan AbstractTableModel -eko metodoak inplementatuko ditugu, taulan zer nahi dugun agertzea jartzeko. Gure taulan Driver -en datuak agertuko dira.

DriverAdapter klasea:

```
package businessLogic;
import java.util.Vector;
import javax.swing.table.AbstractTableModel;
import domain.Driver;
import domain.Ride;
public class DriverAdapter extends AbstractTableModel {
	private Driver d;
	public DriverAdapter(Driver d) {
		this.d = d;
	}

	@Override
	public int getRowCount() {
		  return d.getCreatedRides().size();
	}
	@Override
	public int getColumnCount() {
		return 5;
	}
	@Override
	public Object getValueAt(int rowIndex, int columnIndex) {
		 Ride da = d.getCreatedRides().get(rowIndex);
		  switch (columnIndex) {
		  case 0:
		   return da.getFrom();
		  case 1:
		   return da.getTo();
		  case 2:
		   return da.getDate();
		  case 3:
		   return da.getnPlaces();
		  case 4:
		   return da.getPrice();
		  default:
		   return null;
		  }
	}
}
```

DriverTable klasea:

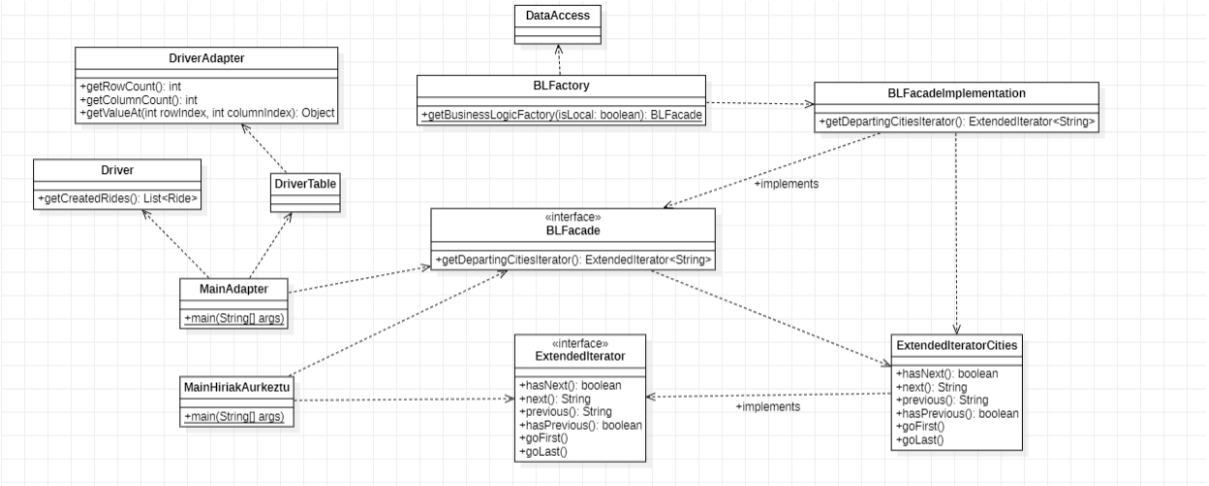
```
package businessLogic;
import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import domain.Driver;
public class DriverTable extends JFrame {
	private Driver driver;
	private JTable tabla;
	public DriverTable(Driver driver) {
		super(driver.getUsername() + "'srides ");
		this.setBounds(100, 100, 700, 200);
		this.driver = driver;
		DriverAdapter adapt = new DriverAdapter(driver);
		tabla = new JTable(adapt);
		tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
		JScrollPane scrollPane = new JScrollPane(tabla);
		getContentPane().add(scrollPane, BorderLayout.CENTER);
	}
}
```

Azkenik main klase bat sortuko dugu, mainAdapter.java JFrame, egin duguna konprobatzeko eta lortu nahi dugun taula ikusteko. Klase honetan FacadeImplementationWS objetu bat eta Driver guztiak jasoko dituen bektore bat sortuko dugu. Gerora beste bektore bat sortu beharko dugu aukeratutako jabearen Driver -ak gordetzeko eta taulan erakusteko. mainAdapter.java klasearen main -ean design atalean garatuko dugun leihoa sortuko du, hasieratze moduan.

Main klasea:

```
package businessLogic;
import domain.Driver;
public class MainAdapter {
	public static void main(String[] args) {
//		the BL is local
		boolean isLocal = true;
		BLFacade blFacade = new BLFactory().getBusinessLogicFactory(isLocal);
		Driver d = blFacade.getDriver("Urtzi");
		DriverTable dt = new DriverTable(d);
		dt.setVisible(true);
	}
}
```

UML-a:


DataAccess
DriverAdapter
+getRowCount(): int
+getColumnCount(): int
+getValueAt(int rowIndex, int columnIndex): Object
BLFactory
+getBusinessLogicFactory(isLocal: boolean): BLFacade
BLFacadeImplementation
+getDepartingCitiesIterator(): ExtendedIterator<String>
Driver
+getCreatedRides(): List<Ride>
DriverTable
+implements
«interface»
BLFacade
+getDepartingCitiesIterator(): ExtendedIterator<String>
MainAdapter
+main(String[] args)
«interface»
ExtendedIterator
+hasNext(): boolean
+next(): String
+previous(): String
+hasPrevious(): boolean
+goFirst()
+goLast()
ExtendedIteratorCities
+hasNext(): boolean
+next(): String
+previous(): String
+hasPrevious(): boolean
+goFirst()
+goLast()
MainHiriakAurkeztu
+main(String[] args)
+implements