GitHub esteka: aitorfontecha/Bets21 (github.com)
Egileak: Iñigo Gil eta Aitor Fontecha

Factory Method Patroia

Factory Method patroia erabiliz, zein negozio logikako inplementazioa erabiliko den erabakitzea ahalbidetzen du. Horretarako, inplementatuko dugu Creator, Product eta ConcreteProduct jokatzen duten klaseen rolak garbi aurkezteko. Honekin lortu nahi da negozio logikako exekuzioa modu lokal edo urrutiko batean egitea FactoryLaunch klaseren bidez.

Horretarako FactoryLaunch klasea BusinessLogic paketean sortu dugu. Honetan createBLFacade metodoaren bitartez aukeratuko da zein eratan exeketutako dugun.

FactoryLaunch.java

```java
package businessLogic;
public class FactoryLaunch {

    public static BLFacade createBLFacade(int num) {
        if(num == 0) {
            ConfigXML c=ConfigXML.getInstance();
            DataAccess da= new
DataAccess(c.getDataBaseOpenMode().equals("initialize"));
            return new BLFacadeImplementation(da);
        }
        if(num == 1) {
            try {
                ConfigXML c=ConfigXML.getInstance();
                String serviceName= "http://"+c.getBusinessLogicNode()
+":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";
                URL url = new URL(serviceName);
                QName qname = new QName("http://businessLogic/",
"BLFacadeImplementationService");
                Service service = Service.create(url, qname);
                return service.getPort(BLFacade.class);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return null;
    }



}
```

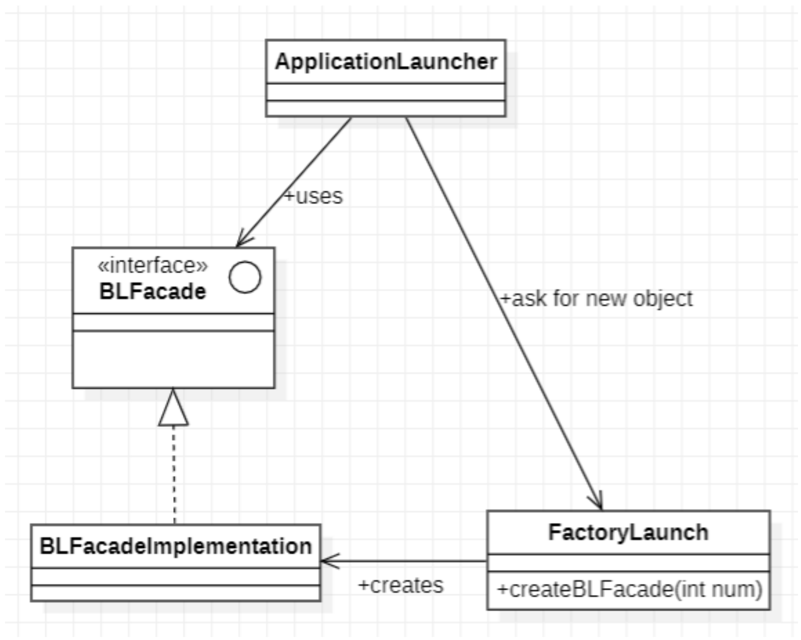
ApplicationLauncher.java

```
import businessLogic.FactoryLaunch;

public class ApplicationLauncher {

      public static void main(String[] args) {

            ConfigXML c = ConfigXML.getInstance();

            System.out.println(c.getLocale());

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

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

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

            try {

                  BLFacade appFacadeInterface;
//
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsClassic
LookAndFeel");
//
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel
");

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

                  if (c.isBusinessLogicLocal()) {
/*
                        // In this option the DataAccess is created by
FacadeImplementationWS
                        // appFacadeInterface=new
BLFacadeImplementation();

                        // In this option, you can parameterize the
DataAccess (e.g. a Mock DataAccess
                        // object)

                        DataAccess da = new
DataAccess(c.getDataBaseOpenMode().equals("initialize"));
                        appFacadeInterface = new
BLFacadeImplementation(da);*/

                        appFacadeInterface =
FactoryLaunch.createBLFacade(0);

                  }

                  else { // If remote

/*                      String serviceName = "http://" +
c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort() + "/ws/"
                                    + c.getBusinessLogicName() + "?wsdl";

                        //URL url = new
URL("http://localhost:9999/ws/ruralHouses?wsdl");
                        URL url = new URL(serviceName);

                        // 1st argument refers to wsdl document above
                        // 2nd argument is service name, refer to wsdl
document above
//                  QName qname = new QName("http://businessLogic/",
"FacadeImplementationWSService");
                        QName qname = new QName("http://businessLogic/",
"BLFacadeImplementationService");

                        Service service = Service.create(url, qname);

                        appFacadeInterface =
service.getPort(BLFacade.class);*/

                        appFacadeInterface =
FactoryLaunch.createBLFacade(1);
                  }
                  /*
                   * if (c.getDataBaseOpenMode().equals("initialize"))
                   * appFacadeInterface.initializeBD();
                   */
                  MainGUI.setBussinessLogic(appFacadeInterface);

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

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

```
            }
            // a.pack();

        }

  }
```

ApplicationLauncher klasean appFacadeInteface sortzeko era aldatu da, orain FactoryLaunch klasean dagoen createBLFacade metodoa erabiltzen da.

Honekin lortu dugu Creator roll-a FactoryLaunch klaseak egitea, honek negozio logikako objektua sortzen du eta. Product rolla BLFacade klasea egiten du orain, FactoryLaunch klasea itzultzen duen objektua delako. ConcreteProduct rolla BLFacadeImplementation egiten du, FactoryLaunch objektua sortzen duelako.



Iterator Patroia

ExtendedIterator interfazea sortu da eta hau erabiliz ExtendedIteratorImplementation klasea inplementatu da. Honetan,iteratorrak behar dituen metodo guztien inplementazioa idatzi da.

Gero BLFacade eta haren inplementazioan List objektuen ordez ExtendedIterator jarri dira, Iterator objektua erabiltzeko getEvents metodoan.

ExtendedIterator.java

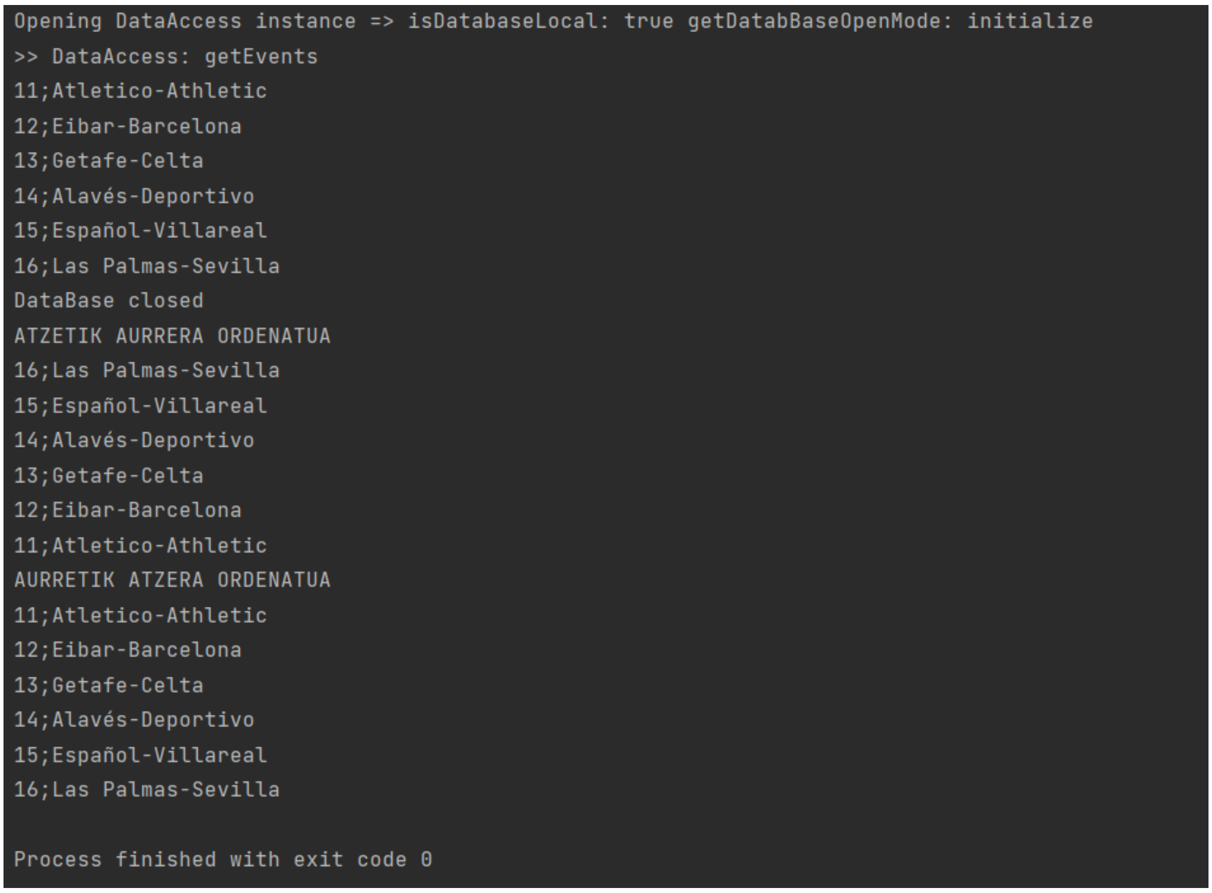
```
package businessLogic;

import java.util.Iterator;
```

```java
public interface ExtendedIterator<Object> extends Iterator {

    // uneko elementua itzultzen du eta aurrekora pasatzen da
    public Object previous();

    // true aurreko elementua existitzen bada.
    public boolean hasPrevious();

    // Lehendabiziko elementuan kokatzen da.
    public void goFirst();

    // Azkeneko elementuan kokatzen da.
    public void goLast();
}
```

ExtendedIteratorImplementation.java

```java
package businessLogic;

public class ExtendedIteratorImplementation<Event> implements
ExtendedIterator<Event> {

    private Vector<Event> vector;
    private int position;

    public ExtendedIteratorImplementation(Vector<Event> pvector) {

        vector = pvector;
        position = 0;
    }

 public Vector<Event> getEvents() {
        return this.vector;
    }

    @Override
    public boolean hasNext() {
        if ((position + 1) < vector.size()) {
            return true;
        } else {
            return false;
        }
    }

    @Override
    public Event next() {
        position++;
        return vector.get(position);
    }

    @Override
    public Event previous() {
        position--;
        return vector.get(position);
    }

    @Override
    public boolean hasPrevious() {
        if ((position - 1) >= 0) {
            return true;
        } else {
            return false;
        }
    }

    @Override
    public void goFirst() {
        position = -1;
    }

    @Override
    public void goLast() {
        position = vector.size();
    }

}
```

Hau guztia egin ondoren, List<Event> objektua erbiltzen dituzten GUI guztiak aldatu egin dira. Ordez ExtendedIteratorImplementation<Event> objektuak jarri dira eta getEvents() metodoa erabili da objektua iteratzeko.


```
Opening DataAccess instance => isDatabaseLocal: true getDatabBaseOpenMode: initialize
>> DataAccess: getEvents
11;Atletico-Athletic
12;Eibar-Barcelona
13;Getafe-Celta
14;Alavés-Deportivo
15;Español-Villareal
16;Las Palmas-Sevilla
DataBase closed
ATZETIK AURRERA ORDENATUA
16;Las Palmas-Sevilla
15;Español-Villareal
14;Alavés-Deportivo
13;Getafe-Celta
12;Eibar-Barcelona
11;Atletico-Athletic
AURRETIK ATZERA ORDENATUA
11;Atletico-Athletic
12;Eibar-Barcelona
13;Getafe-Celta
14;Alavés-Deportivo
15;Español-Villareal
16;Las Palmas-Sevilla

Process finished with exit code 0
```

Adapter Patroia

Patroi hau inplementatuz edozein pertsonaren mugimenduak ikusi ahalko ditugu klase bakar batekin, berdin izango da Bezero edo Administratzaile den. Adaptadorearekin bien apostuak bistaratuko dira.

BezeroaAdapter.java

```
package businessLogic;

public class BezeroaAdapter extends AbstractTableModel {

    private List<Apostua> apostuak;
    private Bezero bezeroa;
    private String[] colNames = new String[] {"Event", "Question",
"Event Date", "Bet(€)"};

    public BezeroaAdapter(Bezero b) {
        this.bezeroa = b;
        this.apostuak = new ArrayList<>(b.getApostuak());
    }
    @Override
    public Object getValueAt(int rowIndex, int colIndex) {
        switch(colIndex) {
            case 0:
                return (Object)
apostuak.get(rowIndex).getPronostikoa().get(0).getGaldera().getEvent().g
etDescription();
            case 1:
                return (Object)
apostuak.get(rowIndex).getPronostikoa().get(0).getGaldera().getQuestion(
);
            case 2:
                return (Object)
apostuak.get(rowIndex).getPronostikoa().get(0).getGaldera().getEvent().g
etEventDate();
            case 3:
                return (Object) apostuak.get(rowIndex).getApustuDiru();
        }
        return null;
    }

    @Override
    public String getColumnName(int col) {
        return colNames[col];
    }

    @Override
    public int getColumnCount() {
        return 4;
    }

    @Override
    public int getRowCount() {
        return apostuak.size();
    }

}
```

GUI berri bat sortu da non adapterraren bitartez lortutako informazioa bistaratuko den, ApostuakIkusiGUI klasea. Eta, GuestOptionsGUI klasetik GUI berri honetara sartzeko botoia eta beharrezkoa den funtzioa gehitu dira.

Domain paketeko ez ditugun aurkitu arazoengatik apustuak ikusteko GUI-ak errore bat ematen du. Nahiz eta errore horiek izan uste dugu kodea era zuzenean egin dugula nahiz eta frogatzeko aukera izan ez dugun.