DISEINU PATROIAK

Taldekideak:

- Jon Ander Jimenez
- Mikel Mujika
- Ibai Heras

Github proiektua:

https://github.com/iByN8/Bet22Lite-master

Factory Method Patroia

Bets aplikazioa hiru mailako arkitektura bat jarraituz diseinatuta dago. Arkitektura honetan, aurkezpenak lokalean edo beste makina batean dagoen negozio logika (hau da, web zerbitzu bat) erabiltzea erabakitzen du. Bi kasuetan, nahiz eta bi objektuek interfaze berdina konpartitu (BLFacade), inplementazioa nabarmenki desberdina da.
Oraingo inplementazioan ApplicationLauncher klaseak zein inplementazioa erabili aukeratzen du. Zehazki main() metodoan, config.xml dagoen isLocal aldagairen balioaren arabera (hau da, c.isBusinessLogicLocal()), appFacadeInterface aldagaiari zein negozio logika objektua (lokala edo urrunekoa) erabili behar den esleitzen dio.

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.

Soluzioa:

Erabil dezagun Factory Method diseinu-eredua BLFacade objektuaren sorkuntza zentralizatzeko. BLFacadeFactory interfaze bat sortuko dugu BLFacade objektu bat itzuliko duen metodo batekin. Gero, bi klase konkretu izango ditugu interfaze hau martxan jartzeko, bata tokiko ezarpenerako (LocalBLFacadeFactory) eta bestea urrutiko (RemoteBLFacadeFactory) ezarpenerako.

ApplicationLauncher kodea aldaketa hau izango litzateke:

```
try {
    BLFacadeFactory factory;
    if (c.isBusinessLogicLocal()) {
        factory = new LocalBLFacadeFactory();
    } else {
        factory = new RemoteBLFacadeFactory();
    }

    BLFacade appFacadeInterface = factory.createBLFacade();
    MainGUI.setBussinessLogic(appFacadeInterface);
```

Hau BLFacadeFactory izango litzateke:

```
interface BLFacadeFactory {
    BLFacade createBLFacade();
}
```

Eta hau bere bi implementazioak, LocalBLFacadeFactory eta RemoteBLFacadeFactory:

```
class LocalBLFacadeFactory implements BLFacadeFactory {
    public BLFacade createBLFacade() {
        DataAccess da = new DataAccess(ConfigXML.getInstance().getDataBaseOpenMode().equals("initialize"));
        return new BLFacadeImplementation(da);
    }
}
```

```
class RemoteBLFacadeFactory implements BLFacadeFactory {
    public BLFacade createBLFacade() {
        ConfigXML c = ConfigXML.getInstance();
        String serviceName = "http://" + c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort() + "/ws/"
                + c.getBusinessLogicName() + "?wsdl";

        try {
            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) {
            throw new RuntimeException("Error creating remote BLFacade: " + e.toString());
        }
    }
}
```

Eta honela geratuko lirateke Creator, Product eta ConcreteProduct jokatzen duten klaseen rolak:

1. **Creator (BLFacadeFactory):** Interfaze edo klase abstraktua da, sorkuntza-metodoa adierazten duena, fabrika-metodo gisa jarduten duena. Emandako kodean, BLFacadeFactory interfaze sortzailea da.
2. **Concrete Product (LocalBLFacadeFactory, RemoteBLFacadeFactory):** Sortzailearen interfazearen inplementazio zehatzak dira. BLFacade metodoaren aplikazioa eskaintzen dute. Kodean, LocalBLFacadeFactory eta RemoteBLFacadeFactory Concrete Product dira.

3. **Product (BLFacade):** Interfaze edo klase abstraktua da, produktu konkretuak inplementatuko dituen metodoak adierazten dituena. Kodean, BLFacade produktu interfazea da.

Iterator patroia

Eskatzen da: Iteratzaile Hedatua inplementatu, eta adibidezko antzeko programa bat inplementatuz, gertaerak aurkeztutako ordenan inprimatu. Jarraian zure aplikazioa aldatu, getEvents() modu berrian erabiltzeko

Hasteko klase bat sortu dut ExtendedIterator deitzen dena, eta hau da implementazioa:

```
package iterator;

import java.util.Iterator;

public interface ExtendedIterator 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();
}
```

Beste klase lagungarri bat egin dut, gero GUI erabiltzen dudalako eta dataAccessen extenderIterator emaitza bueltatzeko, klase honek extendedIterator inplementatuko du. Hau da kodea:

```
package iterator;

import java.util.Vector;

public class ExtendedIteratorEvents implements ExtendedIterator{
    private Vector<Event> events;
    private int pos;
    public ExtendedIteratorEvents(Vector<Event> vector) {
        events = vector;
        pos = 0;
    }
    public Vector<Event> getEvents() {
        return events;
    }
    public void setEvents(Vector<Event> events) {
        this.events = events;
    }
    public Event previous() {
        Event event =events.get(pos);
        pos -=1;
        return event;
    }
    public boolean hasPrevious() {
        return (pos>0);
    }
    public void goFirst() {
        pos = 0;
    }
    public void goLast() {
        pos = events.size()-1;
    }
    @Override
    public boolean hasNext() {
        return pos < events.size();
    }
    @Override
    public Event next() {
        // TODO Auto-generated method stub
        Event event =events.get(pos);
        pos += 1;
        return event;
    }
}
```

Ondoren BLFacade-ren interfazean getEvents metodoaren emaitzaren formatua ExtendedIterator izango da eta ez Vector<Events>. Honek suposatzen du BLFacadeImplementation eta dataAccess klasean getEvents-ren parametro aldatu behar dela. Hau izango litzateke BLFacadeImplementation getEvents metodoaren emaitza:

```java
@WebMethod
public ExtendedIterator getEvents(Date date)  {
    dbManager.open(false);
    ExtendedIterator  events=dbManager.getEvents(date);
    dbManager.close();
    return  events;
}
```

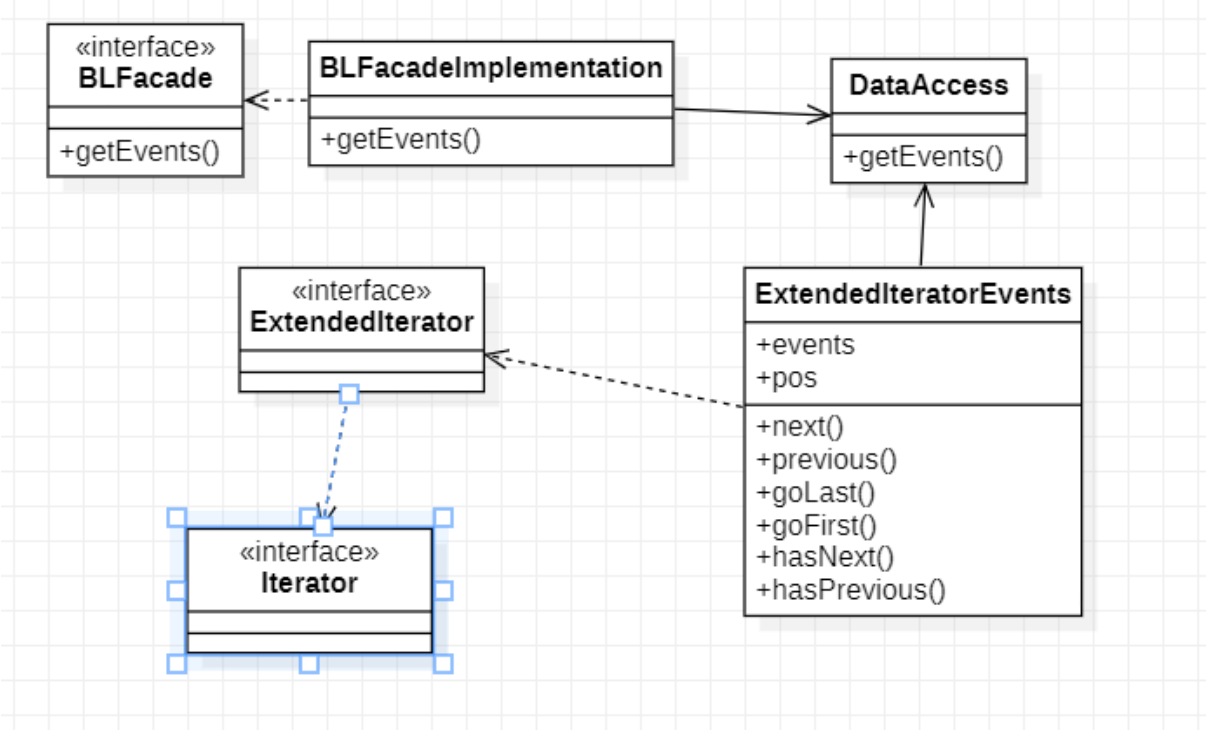
Aldaketak hauek egin ondoren main programa bat eskatzen da nun iterator hau erabiltzen den, nik egindako inplementazioa hau da:

```java
package iterator;

import java.util.Calendar;

public class MainIterator {

    public static void main(String[] args) {
        BLFacadeFactory factory;
        factory = new LocalBLFacadeFactory();

        BLFacade facadeInterface = factory.createBLFacade();

        Calendar today = Calendar.getInstance();//

        int month = today.get(Calendar.MONTH);
        month += 1;
        int year = today.get(Calendar.YEAR);
        if (month == 12) {
            month = 0;
            year += 1;
        }

        ExtendedIterator i = facadeInterface.getEvents(UtilDate.newDate(year, month, 17));
        Event ev;
        i.goLast();
        System.out.println("-------------------------------------------------");
        while (i.hasPrevious()) {
            ev = (Event) i.previous();
            System.out.println(ev.toString());
        }
        System.out.println("----------------ORAIN HASIERATIK HASITA----------------");
        i.goFirst();
        while (i.hasNext()) {
            ev = (Event) i.next();
            System.out.println(ev.toString());
        }
    }
}
```

Exekuzio honen emaitza hau izango litzateke:

```
-------------------------------------------------
27;Djokovic-Federer
24;Miami Heat-Chicago Bulls
23;Atlanta Hawks-Houston Rockets
22;LA Lakers-Phoenix Suns
10;Betis-Real Madrid
9;Real Sociedad-Levante
8;Girona-Leganes
7;Malaga-Valencia
6;Las Palmas-Sevilla
5;Espanol-Villareal
4;Alaves-Deportivo
3;Getafe-Celta
2;Eibar-Barcelona
----------------ORAIN HASIERATIK HASITA----------------
1;Atletico-Athletic
2;Eibar-Barcelona
3;Getafe-Celta
4;Alaves-Deportivo
5;Espanol-Villareal
6;Las Palmas-Sevilla
7;Malaga-Valencia
8;Girona-Leganes
9;Real Sociedad-Levante
10;Betis-Real Madrid
22;LA Lakers-Phoenix Suns
23;Atlanta Hawks-Houston Rockets
24;Miami Heat-Chicago Bulls
27;Djokovic-Federer
```

Azkenik uml-a aldatu beharko genuke. Bakarrik getEvents metodoa jarri dut aldatu dudan bakarra izan delako.



Adapter Patroia

Eskatzen da: JTable batean Erabiltzaile batek egin dituen Apustu guztien informazioa aurkezten duen leiho berri bat sortu. Ohar: Ezin da Erabiltzaile klasea aldatu. Diseina eta inplementatu ebazpena.

Soluzioa:

Lehenengo eta behin MainGUI-an botoi berri bat sortu dugu:

```
private JButton getBtnNewButton() {
    if (btnNewButton == null) {
        btnNewButton = new JButton(ResourceBundle.getBundle("Etiquetas").getString("MainGUI.btnNewButton.text")); //$NON-NLS-1$ //$NON-NLS-2$
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Registered r = new Registered("MaiteUrreta","1234",33243);
                BLFacade facade = MainGUI.getBusinessLogic();
                facade.findUser(r);
                JFrame a = new ApustuakErakutsiGUI(r.getApustuAnitzak());
                a.setVisible(true);
            }
        });
    }
    return btnNewButton;
}
// @jve:decl-index=0:visual-constraint="0,0"
```

Botoi honen bitartez beste interfaze berri batera bidaliko gaitu. Interfaze horrek aurrerako erkutsiko dugun adapterra erabiliko du guk lortu nahi dugun taula sortzeko.

Interfaze berria, ApustuakErakutsiGUI honako hau da:

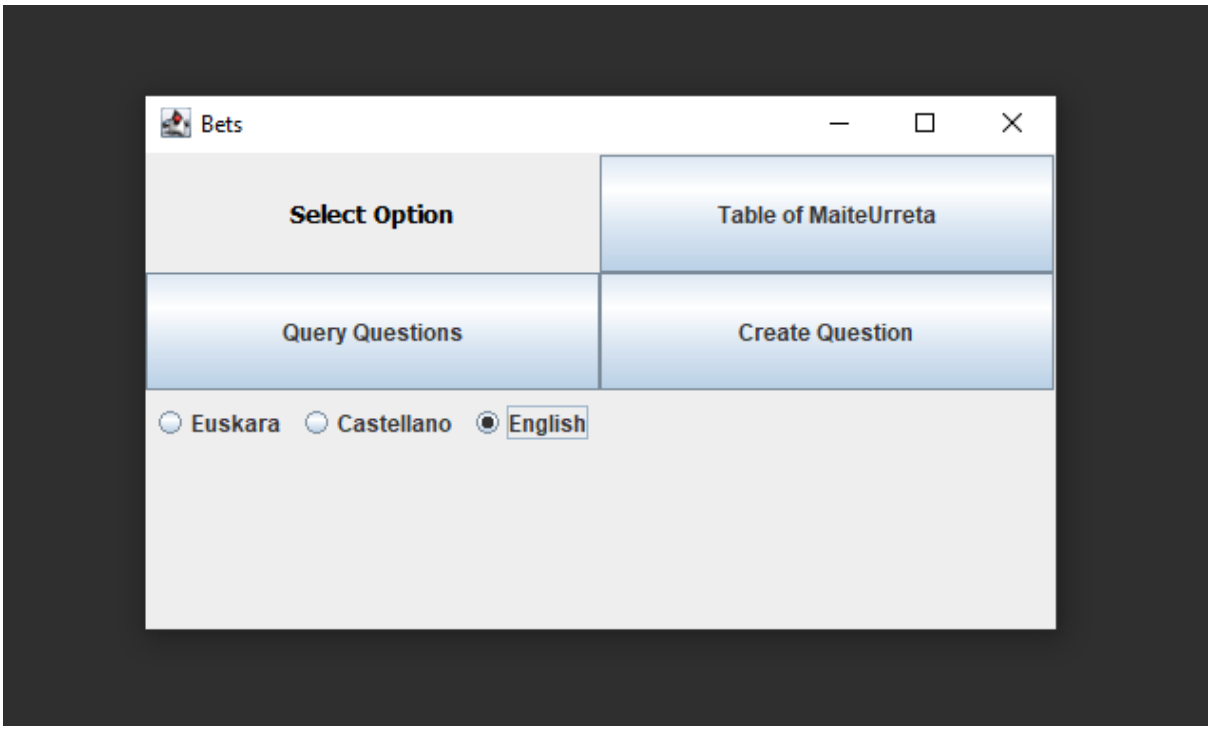
```
package gui;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import businessLogic.BLFacadeImplementation;
import domain.Registered;
import domain.UserAdapter;
import domain.ApustuAnitza;

import java.awt.BorderLayout;
import java.util.ArrayList;
import java.util.List;

public class ApustuakErakutsiGUI extends JFrame {
      private JTable userTable;
        private UserAdapter userAdapter;

        public ApustuakErakutsiGUI(List<ApustuAnitza> listaUsuarios) {
            // Configurar la ventana
            setTitle("Egindako apustuak:");
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setSize(500, 300);

            userAdapter = new UserAdapter(listaUsuarios);
            userTable = new JTable(userAdapter);
            JScrollPane scrollPane = new JScrollPane(userTable);
            getContentPane().add(scrollPane);
            setVisible(true);
        }


}
```

Eskatutakoa lortzeko, UserAdapter izeneko klase berri oso bat sortu dugu. Klase honetan erabiltzaileak lortu ditzazkegun apustuen patroi bat sortzen dugu. Bertako informazioa atera eta taula batean txertatzean dugu.

Gure patroia honela geratuko litzateke:

```java
package domain;

import java.util.List;
import javax.swing.table.AbstractTableModel;

public class UserAdapter extends AbstractTableModel{

    private String[] columnNames = {"Event", "Question", "Event Date", "Bet" };
    private List<ApustuAnitza> apustuList;

    // Constructor to initialize the list of Registered users
    public UserAdapter(List<ApustuAnitza> userList) {
        this.apustuList = apustuList;
    }

    public int getRowCount() {
        return apustuList.size();
    }

    public int getColumnCount() {
        return 4;
    }

    public Object getValueAt(int row, int col) {
        ApustuAnitza ap = apustuList.get(row);

        for (Apustua a : ap.getApustuak()) {
        switch (col) {
            case 0:
                return a.getKuota().getQuestion().getEvent();
            case 1:
                return a.getKuota().getQuestion();
            case 2:
                return a.getKuota().getQuestion().getEvent().getEventDate();
            case 3:
                return a.getApustuAnitza().getBalioa();
            default:
                return null;
        }
    }
        return null;
   }

    public String getColumnName(int col) {
        return columnNames[col];
    }
}
```

Beraz, guk MainGUI-a exekutatzerakoan botoi berri bat egertuko zaigu.



Behin botoi hori sakatzerakoan, beste leiho berri bat agertuko zaigu non MaiteUrreta-ren apustu guztien informazioa agertuko zaigun. Hau dena adapter patroiaren bitartez lortuta.

UML: