

DISEINU PATROIAK

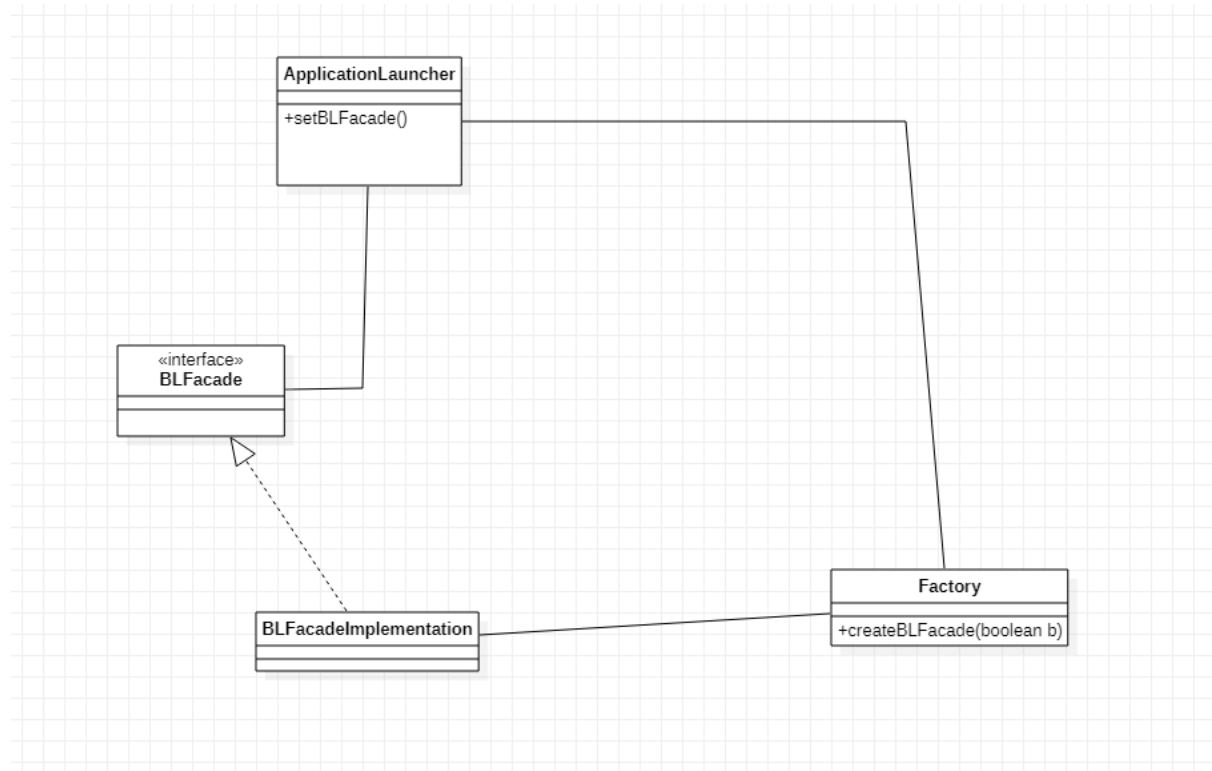


AURKIBIDEA

FACTORY METHOD PATROIA	3
UML	3
KODEAREN AZALPENA	3
ITERATOR PATROIA	6
UML	6
KODEAREN AZALPENA	6
EXEKUZIO IRUDIA	7
ADAPTER PATROIA	8
UML	8
KODEAREN AZALPENA	8
EXEKUZIO IRUDIA	10
GITHUB ESTEKA	11

FACTORY METHOD PATROIA

UML



KODEAREN AZALPENA

Aplikazioan, posible da negozio logika lokalean, edo zerbitzari batean exekutatzea. Bata edo bestea martxan jartzeaz arduratuko dena Factory klasea izango da, Factory patroian eraikitzaile rola jokatzeko duen klasea.

Beraz, esanda bezala Factory izeneko klase bat sortu dugu, boolean baten arabera lokalean edo zerbitzarian exekutatuko duena aplikazioa. Honetarako, noski, ApplicationLauncher klasean hainbat aldaketa egin behar izan ditugu, izan ere, orain arte ApplicationLauncher klasea arduratzen zen sorkuntzaz. (**Application Launcher da bezeroaren rola jokatzeko duena**):

1. Factory klaseko atributua sortu:

```
public class ApplicationLauncher {  
    private static Factory logicFactory = new Factory();  
}
```

2. Factory klaseari deitu BLFacade klaseko objektua sortzeko (pasatutako parametro bakarra, lokalean edo zerbitzarian exekutatu behar den adierazten duena da):

```
try {  
  
    BLFacade appFacadeInterface;  
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel");  
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");  
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");  
  
    appFacadeInterface = logicFactory.createBLFacade(c.isBusinessLogicLocal());  
}
```

Factory klasearen inplementazioa, berriz, honako hau da:

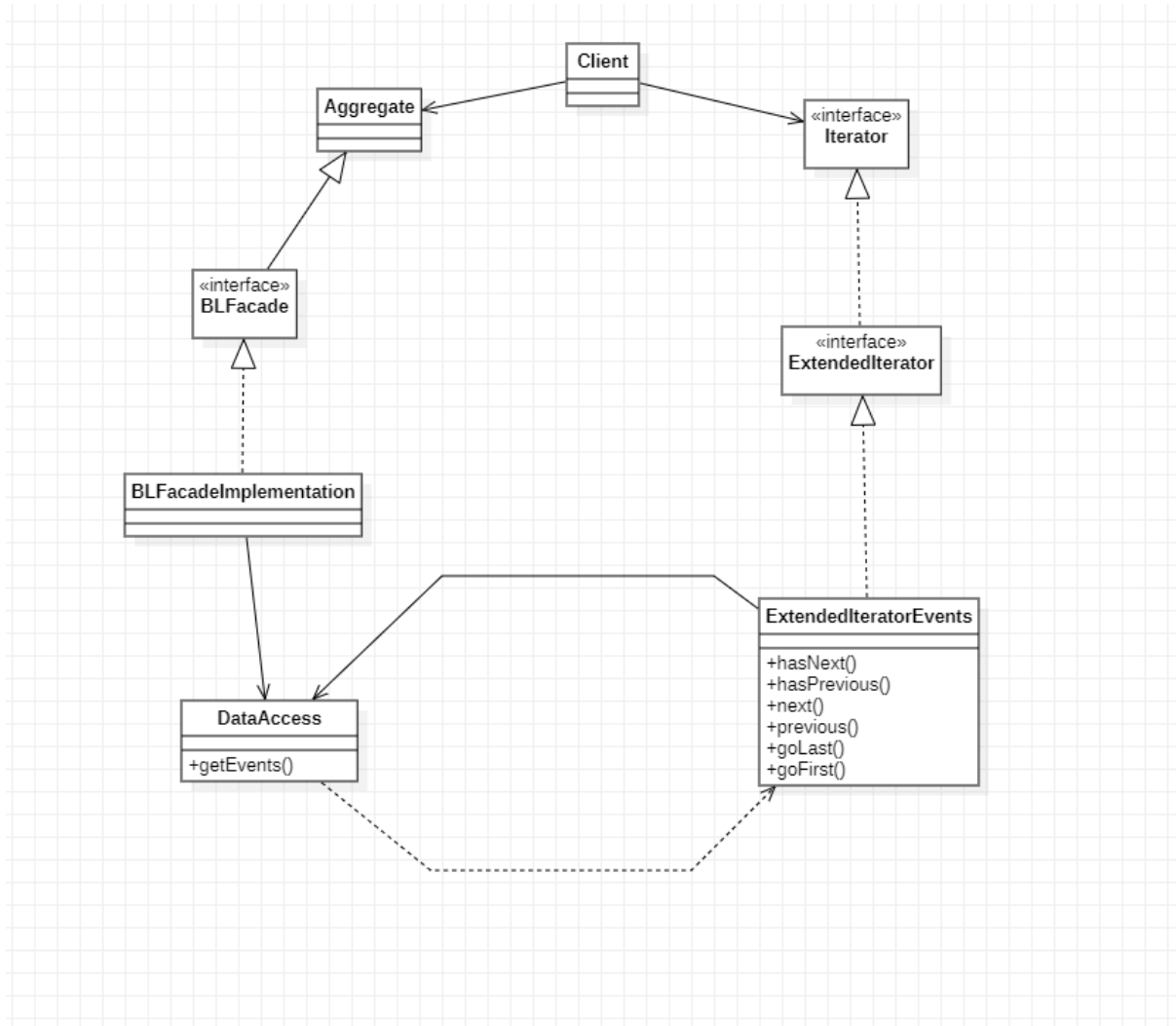
```
package businessLogic;  
  
import java.net.MalformedURLException;  
  
public class Factory {  
  
    public BLFacade createBLFacade(boolean isLocal) {  
  
        ConfigXML c=ConfigXML.getInstance();  
  
        if(isLocal) {  
            //In this option the DataAccess is created by FacadeImplementationWS  
            //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"));  
            return new BLFacadeImplementation(da);  
        }  
        else {  
            String serviceName= "http://"+c.getBusinessLogicNode() +":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";  
  
            //URL url = new URL("http://localhost:9999/ws/ruralHouses?wsdl");  
            URL url=null;  
            try {  
                url = new URL(serviceName);  
            } catch (MalformedURLException e) {  
                // TODO Auto-generated catch block  
                e.printStackTrace();  
            }  
  
            //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);  
  
            return service.getPort(BLFacade.class);  
        }  
    }  
}
```

Honela, Factory klasea arduratuko da negozio logikako objektua sortzeaz, lokalean edo zerbitzarian, ApplicationLauncherrek pasatuko boolearraren arabera. True bada, lokalean, bestela zerbitzarian.

BLFacade interfazea inplementatzen duten klaseak ConcreteProduct rola jokatzen dutenak izango dira, eta BLFacade interfazeak, berriz, Product.

ITERATOR PATROIA

UML



KODEAREN AZALPENA

Iterator patroiak bi atal ditu, batak Iterator interfazea inplementatzen du, eta bestea Aggregate.

Iterator inplementatzen duen ataletik hasita, ondorengoak egin ditugu:

Lehenengo, ExtendedIterator interfazea sortu dugu, hainbat metodo berri gehitzen dituen (previous(), hasPrevious() ...):

```

package iterator;

import java.util.Iterator;

public interface ExtendedIterator extends Iterator {
    public void setEvents(Vector<Event> events);
    //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();
}

```

Ondoren, ExtendedIteratorEvents klasea sortu dugu, ExtendedIterator interfazea inplementatzen duena, eta gertaeren Vector bat korritzeko erabiliko den iteradore konkretua izango dena:

```

package iterator;

import java.util.Vector;

public class ExtendedIteratorEvents implements ExtendedIterator{

    private Vector<Event> events;
    private int position;

    public ExtendedIteratorEvents(Vector<Event> vector) {
        events = vector;
        position = 0;
    }

    public Vector<Event> getEvents() {
        return events;
    }
    public void setEvents(Vector<Event> events) {
        this.events = events;
    }

    //uneko elementua itzultzen du eta aurrekora pasatzen da
    public Event previous() {
        Event event =events.get(position);
        position = position - 1;
        return event;
    }
    //true aurreko elementua existitzen bada.
    public boolean hasPrevious() {
        return (position>0);
    }
    //lehendabiziko elementuan kokatzen da.
    public void goFirst() {
        position = 0;
    }
    //Azkeneko elementuan kokatzen da.
    public void goLast() {
        position = events.size()-1;
    }
    @Override
    public boolean hasNext() {
        // TODO Auto-generated method stub
        return position < events.size();
    }

    @Override
    public boolean hasNext() {
        // TODO Auto-generated method stub
        return position < events.size();
    }
    @Override
    public Event next() {
        // TODO Auto-generated method stub
        Event event =events.get(position);
        position = position + 1;
        return event;
    }
}

```

Aggregate aldean, berriz, negozio logika izango da Iteratorra sortzeaz arduratuko dena `getEvents()` metodoari dei egitean, lortutako gertaerekin Iteradorea sortuz:

Interfazean, `getEvents` metodoaren signatura aldatuko da:

```
/**
 * This method retrieves the events of a given date
 *
 * @param date in which events are retrieved
 * @return collection of events
 */
@WebMethod public ExtendedIterator getEvents(Date date);
```

BLFacadeImplementationen kasuan:

```
/**
 * This method invokes the data access to retrieve the events of a given date
 *
 * @param date in which events are retrieved
 * @return collection of events
 */
@WebMethod
public ExtendedIterator getEvents(Date date) {
    dbManager.open(false);
    ExtendedIterator events=dbManager.getEvents(date);
    dbManager.close();
    return events;
}
```

Azkenik, `DataAccess` klasean, datu-basetik lortutako gertaerekin iteradorea sortuko da:

```
/**
 * This method retrieves from the database the events of a given date
 *
 * @param date in which events are retrieved
 * @return collection of events
 */
public ExtendedIterator getEvents(Date date) {
    System.out.println(">> DataAccess: getEvents");
    Vector<Event> res = new Vector<Event>();
    TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?1",Event.class);
    query.setParameter(1, date);
    List<Event> events = query.getResultList();
    for (Event ev:events){
        System.out.println(ev.toString());
        res.add(ev);
    }
    ExtendedIteratorEvents iterator = new ExtendedIteratorEvents(res);
    return iterator;
}
```


Amaitzeko, egindakoa probatzeko main klase bat sortu dugu:

```
package iterator;

import java.util.Calendar;

public class main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        boolean isLocal=true;

        //Facade objektua lortu lehendabiziko ariketa erabiliz
        Factory f = new Factory();
        BLFacade facadeInterface;
        facadeInterface = f.createBLFacade(isLocal);
        //BLFacade facadeInterface=.....
        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));

        // Vector events = facadeInterface.getEvents(UtilDate.newDate(year,month,17));
        //
        // ExtendedIterator i = (ExtendedIterator) new EventList(events).getIterator();

        System.out.println("=====\n\n\n");
        System.out.println("=====");

        Event ev;
        i.goLast();
        while (i.hasPrevious()){
            ev=(Event)i.previous();
            System.out.println(ev.toString());
        }
        //Nahiz eta suposatu hasierara ailegatu garela, eragiketa egiten dugu.
        i.goFirst();
        while (i.hasNext()){
            ev=(Event)i.next();
            System.out.println(ev.toString());
        }
    }
}
```

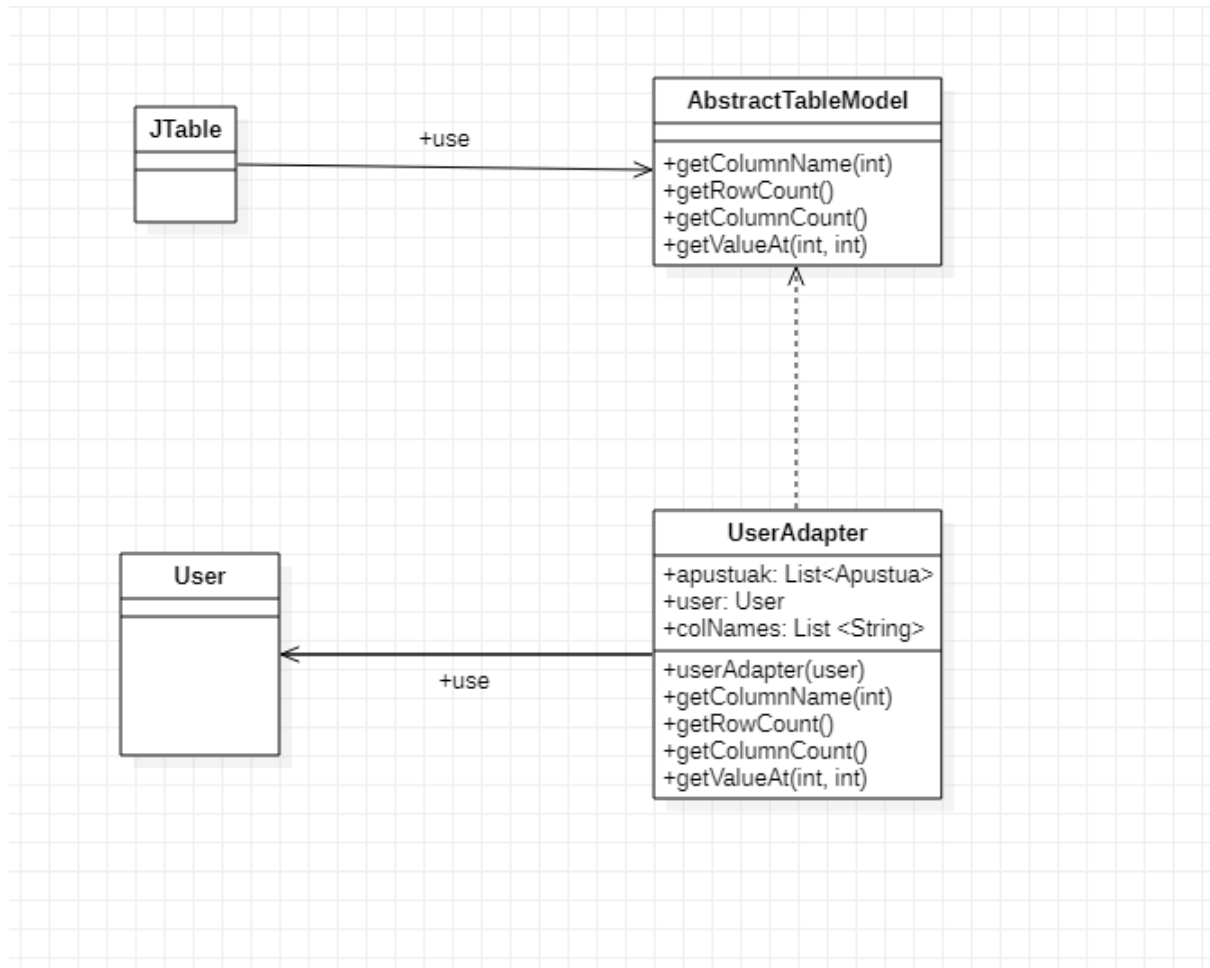
EXEKUZIO IRUDIA

Gertaerak atzetik aurrera eta aurretik atzera korrituz gero...

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
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

ADAPTER PATROIA

UML



KODEAREN AZALPENA

Azkenik, Adapter patroia inplementatu dugu.

JTable batean ezin da zuzenean User baten informazioa erakutsi, JTable bati AbstractModel klaseko objektu bat pasa behar zaio eta. Hori dela eta, UserAdapter klasea sortu dugu, AbstractModel hedatzen duena, eta Userren informazioa jaso, hau egokituz. Honetarako kodea:

```

public class UserAdapter extends AbstractTableModel{
    private List<Apustua> apustuak=new ArrayList();
    private User user;
    private String[] colNames = {"Event", "Question", "EventDate", "Bet(€)"};

    public UserAdapter(User u) {
        Registered r = (Registered) u;
        this.user = u;
        for(ApustuAnitza aa: r.getApustuAnitzak()) {
            for(Apustua a: aa.getApustuak()) {
                apustuak.add(a);
            }
        }
    }

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

    @Override
    public int getRowCount() {
        // TODO Auto-generated method stub
        return apustuak.size();
    }

    @Override
    public int getColumnCount() {
        // TODO Auto-generated method stub
        return 4;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        // TODO Auto-generated method stub
        Apustua a = apustuak.get(rowIndex);
        if(columnIndex==0) {
            return a.getKuota().getQuestion().getEvent();
        }
        else if(columnIndex==1) {
            return a.getKuota().getQuestion();
        }
        else if(columnIndex==2) {
            return a.getApustuAnitza().getData();
        }
        else {
            return a.getApustuAnitza().getBalioa();
        }
    }
}

```

UserAdapter klaseak, AbstractTableModeleko metodoak implementatzen ditu (batzuk aldatu), eta User bat jaso bere informazioa egokitzeko. Metodo esanguratsu bakarra getValueAt() da. Hau arduratuko da JTableko posizio bakoitzean zer kokatu behar den zehazteaz. Hau errazago egiteko, Apustu guztiak Vector batean gorde ditugu, aurretik erabilitako datu egitura zertxobait konplikatuagoa baitzen (apustu anitzak, azken horietako bakoitzak apustu ugari izanik). Modu honetara, errenkada bakoitzean apustu bat eta bere informazioa ipintzea oso erraza da.

Ondoren, JTable hori erakusteko klase bat sortu dugu (interfaze grafiko bat JTable bat eta JScrollPane batekin).

Ondo doan probatzeko main klase bat sortu dugu:

```
package adapter;

import java.awt.BorderLayout;

public class Main {

    public static void main(String[] args) {
        try {
            ConfigXML c=ConfigXML.getInstance();

            //Facade objektua lortu lehendabiziko ariketa erabiliz
            Factory f = new Factory();
            BLFacade facadeInterface;
            facadeInterface = f.createBLFacade(c.isBusinessLogicLocal());
            Registered u = new Registered("mikel","123",123);

            User user = facadeInterface.findUser(u);
            Registered r = (Registered)user;

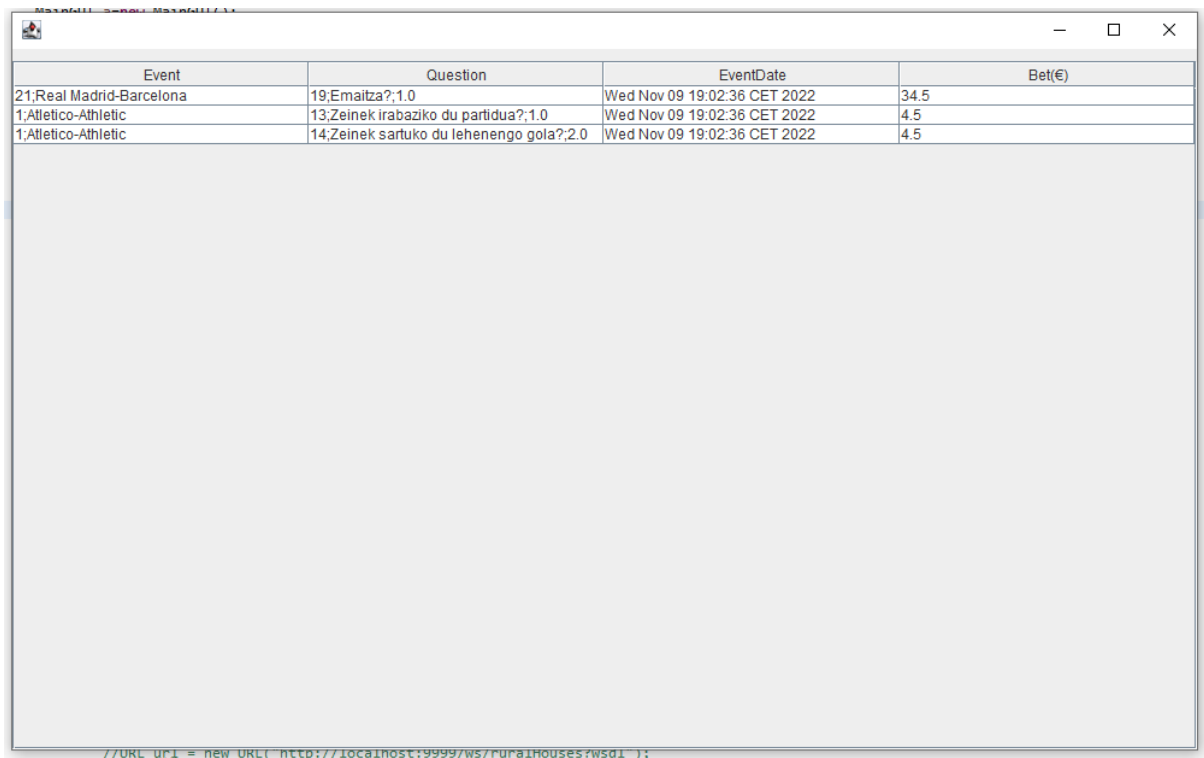
            UserAdapter ua = new UserAdapter(r);
            JTable table = new JTable(ua);

            ApustuakIkusiGUI gui = new ApustuakIkusiGUI(table);
            gui.setVisible(true);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Main klase honetan JTable bat pasatzen diogu ApustuakIkusiGUI interfazeari (aurretik aipatutakoa). JTable honi Useraren informazioa pasatzeaz arduratzen den klasea esandako UserAdapter klasea da, ondoren JTablea sortzeko erabiltzen den AbstractModel klaseko objektua izanik.

EXEKUZIO IRUDIA



The screenshot shows a Java Swing window titled "MainFrame - Real Madrid / 11". The window contains a table with the following data:

Event	Question	EventDate	Bet(€)
21;Real Madrid-Barcelona	19;Ezaitza?;1.0	Wed Nov 09 19:02:36 CET 2022	34.5
1;Atletico-Athletic	13;Zeinek irabaziko du partidua?;1.0	Wed Nov 09 19:02:36 CET 2022	4.5
1;Atletico-Athletic	14;Zeinek sartuko du lehenengo gola?;2.0	Wed Nov 09 19:02:36 CET 2022	4.5

Below the table, there is a large empty rectangular area. At the bottom of the window, the following code is visible:

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

GITHUB ESTEKA

<https://github.com/AlexAmena/BetCompleted22>