2022ko azaroaren 13a

SOFTWARE INGENIARITZA 2

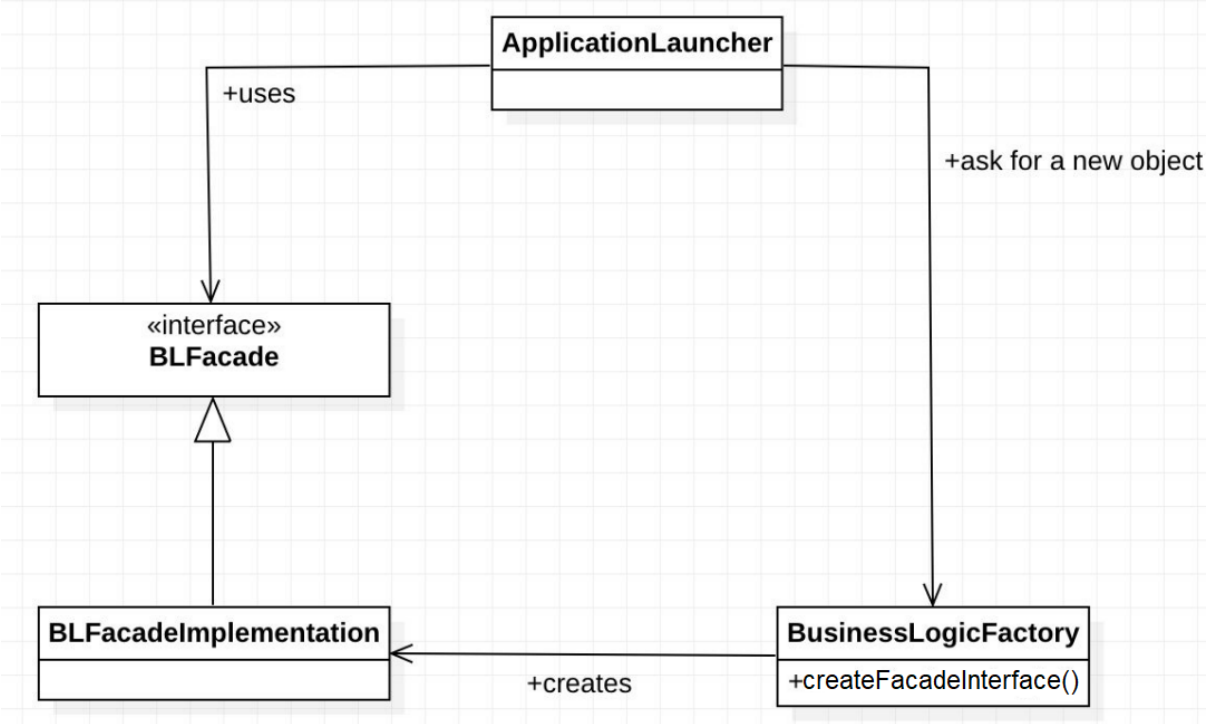
Diseinu patroiak BETS proiektuan

GitHub: https://github.com/anderrubio/patterns.git

Ander Rubio
Ainhitze Ituarte
Aimar Mancisidor

Factory Method Patroia

- UML Diagrama:



Ikusten denez, orain *ApplicationLauncher*-ek ez du *BLFacade* motako objektua sortzen, baizik eta *BusinessLogicFactory* klaseari eskatzen dio bat sortzeko.

Creator rol-a *BusinessFactory* klaseak betetzen du, hau baita objektuak sortzeko arduraduna.

Product rol-a *BLFacade* interfazeak betetzen du, negozio logikaren objektu denek interfaze hori inplementatzen dutelako.

ConcreteProduct rol-a *BLFacadeImplementation* klaseak betetzen du, interfazearen inplementazio bat delako, hau da, *BLFacadeImplementation* guztiak *BLFacade*-ak dira, gainera *BusinessLogicFactory* klaseak mota honetako objektuak sortzen ditu.

- Aldatu duzun kodea, lerro garrantzitsuenak azalduz.

BusinessLogicFactory klasea sortu dugu:

```
1  package businesslogic;
2
3  import java.net.URL;
10
11 public class BusinessLogicFactory {
12
13     public BLFacade createFacadeInterface(boolean isLocal, ConfigXML c) {
14         if(isLocal) {
15             DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
16             BLFacadeImplementation appFacadeInterface=new BLFacadeImplementation(da);
17             return appFacadeInterface;
18         }
19         else {
20            try{
21                String serviceName= "http://"+c.getBusinessLogicNode() +":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";
22                 URL url = new URL(serviceName);
23                 QName qname = new QName("http://businesslogic/", "BLFacadeImplementationService");
24                 Service service = Service.create(url, qname);
25                 BLFacade appFacadeInterface = service.getPort(BLFacade.class);
26                 return appFacadeInterface;
27            }
28            catch(Exception e) {
29                return null;
30            }
31         }
32     }
33 }
```

ApplicationLaucher-ek *createFacadeInterface* metodoari deituko dio, negozio logika lokala edo urrunekoa nahi duen esaten, eta honek mota horretako negozio logikako objektu bat itzultzen du *appFacadeInterface* aldagaian.

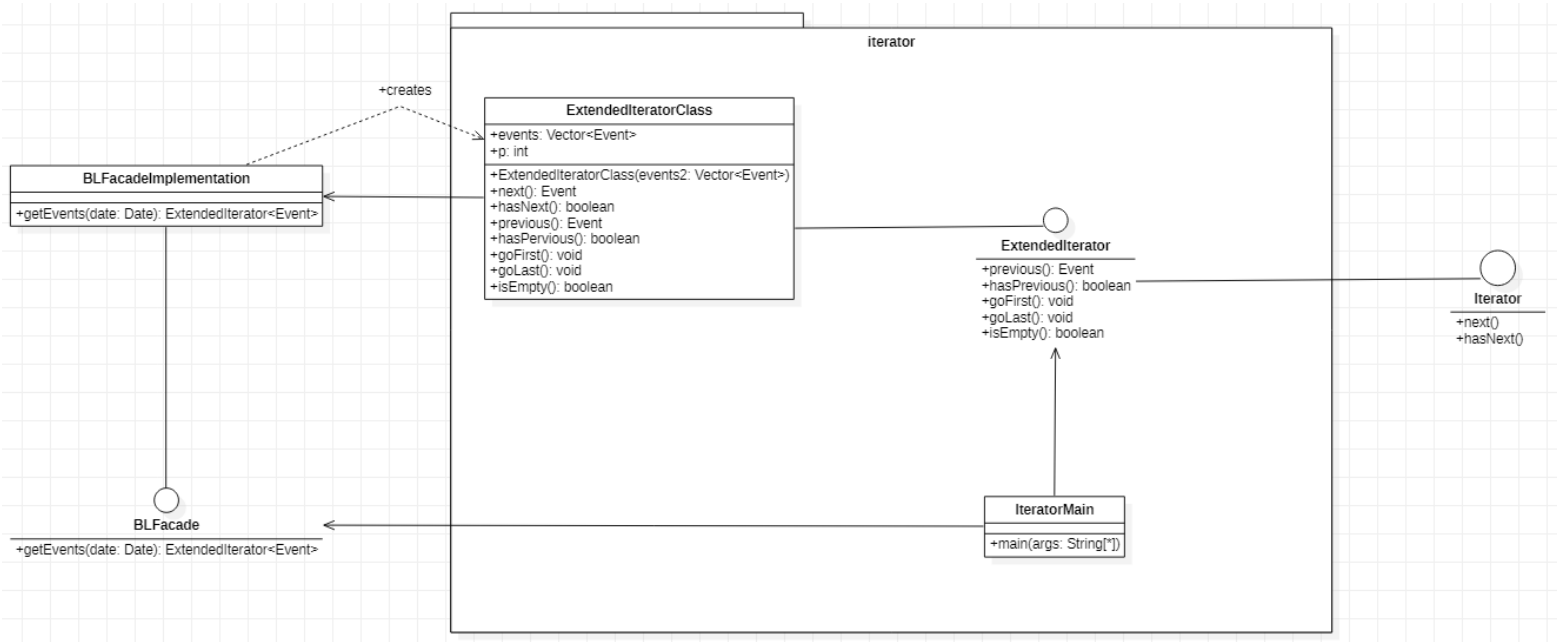
ApplicationLauncher klasea aldatu dugu:

```
17 public class ApplicationLauncher {
18
19
20
21     public static void main(String[] args) {
22
23         ConfigXML c=ConfigXML.getInstance();
24
25         System.out.println(c.getLocale());
26
27         Locale.setDefault(new Locale(c.getLocale()));
28
29         System.out.println("Locale: "+Locale.getDefault());
30         BusinessLogicFactory ff = new BusinessLogicFactory();
31
32         MainGUI a=new MainGUI();
33         a.setVisible(false);
34
35         MainUserGUI b = new MainUserGUI();
36         b.setVisible(true);
37
38
39         BLFacade appFacadeInterface;
40         try{
41             UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
42             appFacadeInterface = ff.createFacadeInterface(c.isBusinessLogicLocal(), c)
43             MainGUI.setBussinessLogic(appFacadeInterface);
44
45         }
46         catch(Exception e){
47             a.jLabelSelectOption.setText("Error: "+e.toString());
48             a.jLabelSelectOption.setForeground(Color.RED);
49
50             System.out.println("Error in ApplicationLauncher: "+e.toString());
51         }
52     }
53
54 }
```

Laukizuzenaren barruan dagoena aldatu dugu. Hor *appFacadeInterface* objektua sortzeko *BusinessLogicFactory*-ri eskatzen dio negozio logikako objektu bat sortzeko berak sortu beharrean eta *MainGUI*-ri esleitzen dio, lehen egiten zuen bezala.

Iterator Patroia

- UML diagrama:



Irudian ikusten den moduan, *package* berri bat sortu dugu, non honakoak gehitu ditugun:

1. *ExtendedIterator* interfazea, enuntziatuan ematen zaiguna eta *Iterator* interfazearen luzapen bat dena (*extends*).

2. *ExtendedIteratorClass* klasea, aurretik aipatutako *ExtendedIterator* interfazea implementatzen duena (*implements*).

3. *IteratorMain* klasea, enuntziatuan ematen zaigun *main* metodoa duena.

Hauetaz gain, *BLFacade* interfazean eta honako hau inplementatzen duen *BLFacadeImplementation* klasean *getEvents* metodoak aldatu egin ditugu, gertaeren bektore bat itzuli ordez, gertaeren *ExtendedIterator* bat itzuli dezaten.

- Aldatu duzun kodea, lerro garrantzitsuenak azalduz

DataAccess klasean *getEvents* metodoa aldatu dugu, gertaeren bektorea *ExtendedIteratorClass* klasearen instantzia batean gehituz eta hau itzuliz. Izan ere, gure helburua metodoak gertaeren *ExtendedIterator* bat itzuli dezan da.

```
791    /**
792     * This method retrieves from the database the events of a given date
793     *
794     * @param date in which events are retrieved
795     * @return collection of events
796     */
797    public ExtendedIterator<Event> getEvents(Date date) {
798        System.out.println(">> DataAccess: getEvents");
799        Vector<Event> res = new Vector<Event>();
800        ExtendedIteratorClass ev = new ExtendedIteratorClass(res);
801        TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?1",Event.class);
802        query.setParameter(1, date);
803        List<Event> events = query.getResultList();
804        for (Event eve:events){
805            System.out.println(eve.toString());
806            res.add(eve);
807        }
808        return ev;
809    }
```

BLFacade interfazean 60.lerroan komentatuta ikusten den lerroa 61.lerroarengatik aldatu dugu. Aurretik aipatu dugun moduan, gertaeren bektore baten beharrean gertaeren *ExtendedIterator* bat itzultzea nahi dugulako.

```
54     /**
55      * This method retrieves the events of a given date
56      *
57      * @param date in which events are retrieved
58      * @return collection of events
59      */
60     //@WebMethod public Vector<Event> getEvents(Date date);
61     @WebMethod public ExtendedIterator<Event> getEvents(Date date);
62
```

BLFacadeImplementation klasean, aurretik ikusi dugun *DataAcces* klaseko getEvents metodoari deitzen diogu eta lortutako *ExtendedIterator*-a itzultzen dugu.

```
103    /**
104     * This method invokes the data access to retrieve the events of a given date
105     *
106     * @param date in which events are retrieved
107     * @return collection of events
108     */
109    @WebMethod
110    public ExtendedIterator<Event> getEvents(Date date)  {
111        dbManager.open(false);
112        ExtendedIterator<Event>  events = dbManager.getEvents(date);
113        dbManager.close();
114        return events;
115    }
```

ExtendedIterator interfazean, enuntziatuan ematen zaigun kodea txertatu dugu. Kode honi *isEmpty()* metodoa ere gehitu diogu, erabilgarria egin zaigulako gero GUI-etan erabiltzeko.

```
package iterator;

import java.util.Iterator;

public interface ExtendedIterator<Object> extends Iterator<Object> {
    //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();

    public boolean isEmpty();
    }

```

ExtendedIteratorClass klasea gertaeren bektore batez eta posizioa aztertzeko *int* batez osatuta dago. Klase honetan, oraintxe ikusitako *ExtendedIterator* interfazeko metodoak inplementatu ditugu aipatutako egitura honetara egokituz.

```
package iterator;

import java.util.Vector;

public class ExtendedIteratorClass implements ExtendedIterator<Event>{
    Vector<Event> events;
    int p = 0;

    public ExtendedIteratorClass(Vector<Event> events2) {
        this.events = events2;
        //this.p = events2.size()-1;
    }

    @Override
    public Event next() {
        Event ev = events.get(p);
        p = p+1;
        return ev;
    }

    @Override
    public boolean hasNext() {
        return p<events.size();
    }

    @Override
    public Event previous() {
        Event ev = events.get(p);
        p = p-1;
        return ev;
    }

    @Override
    public boolean hasPrevious() {
        return p>0;
    }

    @Override
    public void goFirst() {
        p = 0;
    }

    @Override
    public void goLast() {
        p = events.size()-1;
    }

    @Override
    public boolean isEmpty() {
        // TODO Auto-generated method stub
        return events.isEmpty();
    }
```

IteratorMain klasean, enuntziatuan ematen zaigun *main* metodoa txertatu dugu.

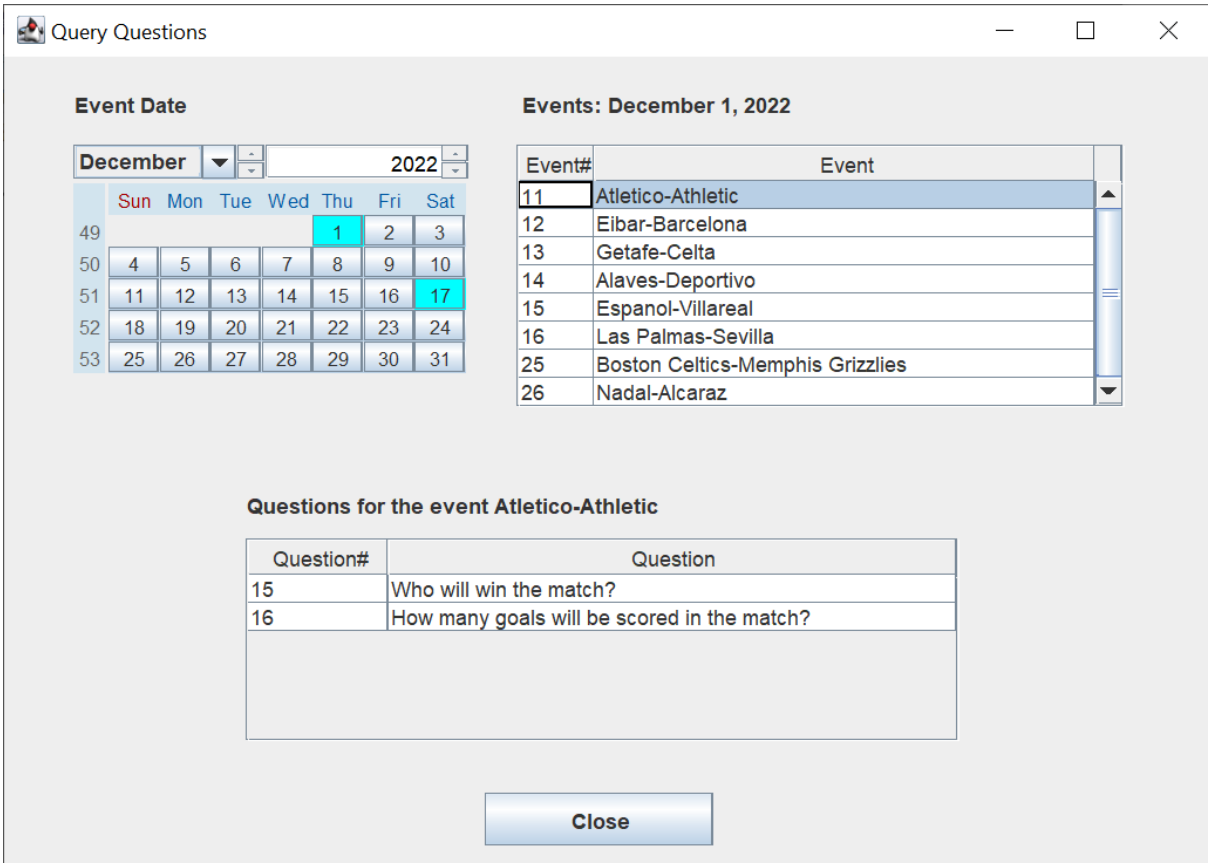
```
package iterator;


import java.text.ParseException;

public class IteratorMain {
    public static void main(String[] args) {
        boolean isLocal=true;
        //Facade objektua lortu lehendabiziko ariketa erabiliz
        BLFacade facadeInterface = (new BusinessLogicFactory()).createFacadeInterface(isLocal, null);
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Date date;

        try {
            date = sdf.parse("09/12/2022");
            ExtendedIterator<Event> i = facadeInterface.getEvents(date);
            Event ev;
            System.out.println("_______");
            System.out.println("Atzetik aurrera");
            i.goLast();
            while (i.hasPrevious()){
                ev=i.previous();
                System.out.println(ev.toString());
            }
            System.out.println();
            System.out.println("_______");
            System.out.println("Aurretik atzera");
            i.goFirst();
            while (i.hasNext()){
                ev=i.next();
                System.out.println(ev.toString());
            }
        }catch (ParseException e1){
            System.out.println("Problems");
        }

    }
}

```

Behin hau eginda, hainbat GUI-tan aldaketa txikiak egin behar izan ditugu, egitura berri honetara egokitzeko, gehienbat *hasNext()* eta *next()* metodoak erabiliz. Hona hemen adibide bezala *FindQuestionsGUI* klasea:

```
try {
    tableModelEvents.setDataVector(null, columnNamesEvents);
    tableModelEvents.setColumnCount(3); // another column added to allocate ev objects

    BLFacade facade=MainGUI.getBusinessLogic();

    ExtendedIterator<Event> events=facade.getEvents(firstDay);

    if (events.isEmpty()) jLabelEvents.setText(ResourceBundle.getBundle("Etiquetas").getString("NoEvents")+ ": "+dateformat1.f
    else jLabelEvents.setText(ResourceBundle.getBundle("Etiquetas").getString("Events")+ ": "+dateformat1.format(calendarAct.g
        while(events.hasNext()) {

        Vector<Object> row = new Vector<Object>();

        System.out.println("Events "+events);

        Event e = events.next();

        row.add(((Event) e).getEventNumber());
        row.add(((Event) e).getDescription());
        row.add(e); // ev object added in order to obtain it with tableModelEvents.getValueAt(i,2)
        tableModelEvents.addRow(row);

    }

    tableEvents.getColumnModel().getColumn(0).setPreferredWidth(25);
    tableEvents.getColumnModel().getColumn(1).setPreferredWidth(268);
    tableEvents.getColumnModel().removeColumn(tableEvents.getColumnModel().getColumn(2)); // not shown in JTable
} catch (Exception e1) {
    e1.printStackTrace();
    jLabelQueries.setText(e1.getMessage());
}
```

- Exekuzioaren irudi bat

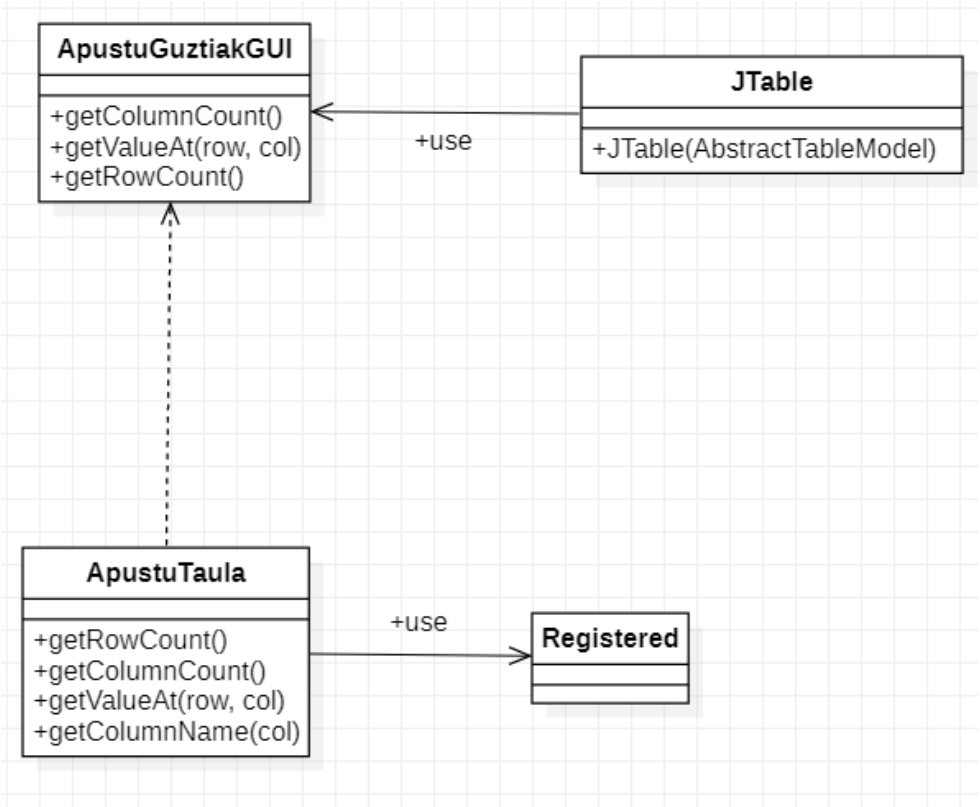
Aldatu dugun *getEvents* metodoa kalendarioa erabiltzen duten GUI-etan erabiltzen da. Beraz, ikus dezagun hauetako baten exekuziaren adibidea. Aurreko atalean *FindQuestionsGUI*-ren inplementazioa ikusi dugunez, honen exekuzioa hartuko dugu adibide moduan:



Datu-basearekin konparazioa egin dugu eta ziurtatu dugu abenduaren 1ean zortzi gertaera horiek zeudela. Gainera, ikusten den bezala, gertaerek ez dituzte haien galderak galdu.

Adapter patroia

- UML Diagrama



Pakete berri bat sortu dugu *adapter* izenekoa eta bertan *ApustuTaula* klasea sortu dugu. *ApustuTaula* klase hau *JTable*-ri emango diogun *Model*-a izango da, beraz, *AbstractTableModel* klasearen luzapen bat izango da (*extends*). *ApustuTaulak Registered* bat behar du (saioa hasita duena) bere Apustuak begiratzeko.

ApustuGuztiakGUI JFrame bat da eta bertan *JTable* bat sortzen dugu. *JTable* horrek *Model* gisa lehen esandako *ApustuTaula* erabiliko du.

- Aldatu duzun kodea, lerro garrantzitsuenak azalduz

ApustuTaula klasea sortu dugu:

```
14 public class ApustuTaula extends AbstractTableModel{
15     /**
16      *
17      */
18     private static final long serialVersionUID = 1L;
19     private String[] columnNames = {"Event","Question","Event Date", "Bet(€)"};
20     private Vector<ApustuAnitza> myBets;
21     public ApustuTaula(Registered r){
22         myBets = r.getApustuAnitzak();
23     }
24     public int getColumnCount() {
25         return columnNames.length;
26     }
27     public int getRowCount() {
28         int size;
29         if (myBets == null) {
30             size = 0;
31         }
32         else {
33             size = myBets.size();
34         }
35         return size;
36     }
37     public Object getValueAt(int row, int col) {
38         Object temp = null;
39         for(int row2=0; row2<myBets.get(row).getApustuak().size(); row2++) {
40             if (col == 0) {
41                 temp = myBets.get(row).getApustuak().get(row2).getKuota().getQuestion().getEvent();
42             }
43             else if (col == 1) {
44                 temp = myBets.get(row).getApustuak().get(row2).getKuota().getQuestion();
45             }
46             else if (col == 2) {
47                 temp = myBets.get(row).getData();
48             }
49             else if (col == 3) {
50                 temp = myBets.get(row).getBalioa();
51             }
52         }
53         return temp;
54
55     }
56     public String getColumnName(int col) {
57         return columnNames[col];
58     }
59 }
```

columnNames aldagaian zutabeek edukiko dituzten izanak zehazten ditugu. Taularen lehen zutabean *Event*-a gordetzen da, bigarrenean *Question*-a, hirugarrenean data eta azkenean balioa. *col* aldagaiak kontrolatzen du zein zutabetan gorde informazioa .

ApustuGuztiakGUI klasea sortu dugu:

```
21 public class ApustuGuztiakGUI extends JFrame {
22     private static final long serialVersionUID = 1L;
23     private JFrame frame;
24     private JTable table;
25     private JFrame thisw;
26     public ApustuGuztiakGUI(User u) {
27         thisw = this;
28         frame = new JFrame();
29         frame.setBounds(100,100,650, 500);
30 
31         BLFacade facade = MainGUI.getBusinessLogic();
32         Registered per = facade.getRegisteredFromUser(u);
33         ApustuTaula model = new ApustuTaula(per);
34 
35 
36         setBounds(100, 100, 896, 642);
37         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38         getContentPane().setLayout(null);
39         table = new JTable(model);
40         JScrollPane scrollPane = new JScrollPane(table);
41         scrollPane.setBounds(200, 210, 610, 480);
42         scrollPane.setPreferredSize(new Dimension(580,480));
43         JPanel panel = new JPanel();
44         panel.setBounds(0, 0, 882, 605);
45         panel.setLayout(null);
46         panel.add(scrollPane);
47         getContentPane().add(panel);
48 
49         JButton atzeraButton = new JButton("Back");
50         atzeraButton.addActionListener(new ActionListener() {
51             public void actionPerformed(ActionEvent e) {
52                 thisw.setVisible(false);
53             }
54         });
55         atzeraButton.setBounds(607, 530, 176, 45);
56         panel.add(atzeraButton);
57 
58     }
```

JTable bat duen GUI arrunt bat da, eta *JTable* hori sortzerakoan *ApustuTaula* pasatzen diogu *model* gisa. Azkenik atzera egiteko botoi bat ere jarri dugu.

*Oharra: *RegisteredGUI*-n “Table of {uneko erabiltzailearen *username*}” izeneko botoia gehitu dugu *ApustuGuztiakGUI*-a irekitzeko. Ez dugu horren kodea jartzen botoi arrunt bat sortzea besterik ez delako, baina exekuzioaren irudian ikusiko duzu.

- Exekuzioaren irudi bat