

Proiektuaren Diseinu Patroiak

Aitor Velaz Nicolas eta Aritz Plazaola Cortabarria

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

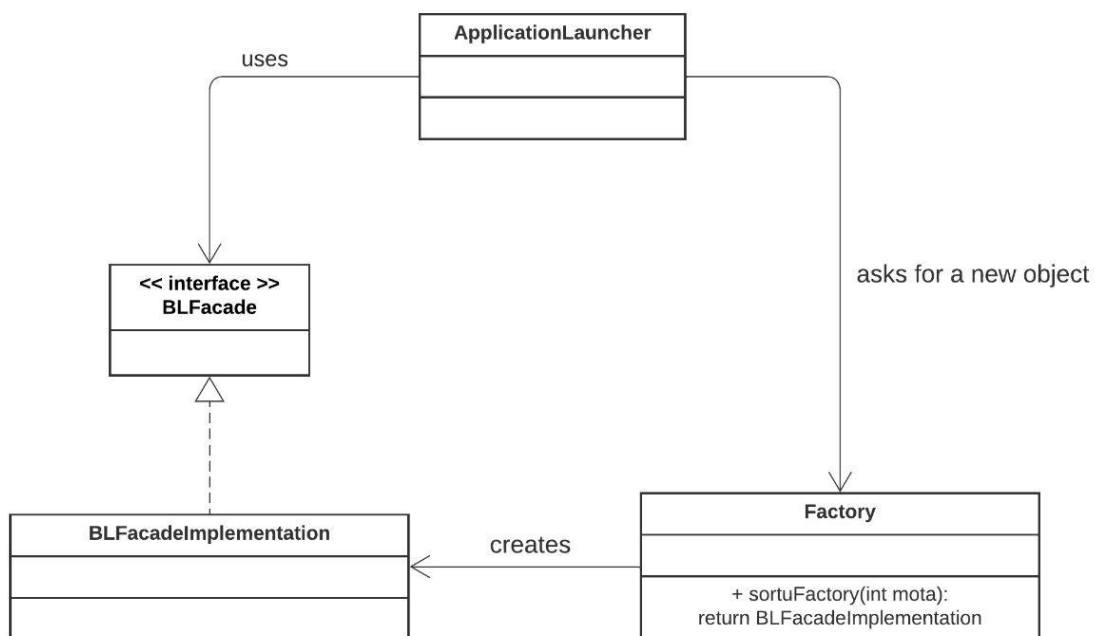
AURKIBIDEA

1.Sarrera.....	2
2.Factory Method Patroia.....	2
3.Iterator Patroia.....	4
4.Adapter Patroia.....	9

1. SARRERA

Dokumentu honetan proiektuan aplikatutako diseinu patroiak aukeztuko ditugu. Proiektu honetan, Bets proiektuan diseinu patroi konkretu batzuk aplikatzea eskatzen da, horretarako beharrezkoak diren luzapen eta aldaketa batzuk eginez.

2. FACTORY PATROIA



Factory Method patroia erabiliko dugu negozio logikako objektuaren lorpena faktoria objektu batean zentralizatuta egoteko, eta aurkezpenak zein negozio logikako implementazio erabili erabaki dezatela. Horretarako, *Creator*, *Product* eta *ConcreteProduct* rolak jokatzen duten klaseak diseinatu eta implementatuko ditugu. Lortu nahi duguna, negozio logikako exekuzioaren erabakia, Factory-ren bidez modu lokal edo urrutiko batean egitea da.

BusinessLogic paketean **Factory** klasea sortuko dugu, bertan *sortuFactory()* metodoa izango dugu, non parametro bezala, zenbaki bat (int) pasatuko diogun. Parametroa 0 bada, exekuzioa lokalean egingo da, aldiz, parametroa 1 bada, urrutiko exekuzioa izango da.

```

9 import configuration.ConfigXML;
10 import dataAccess.DataAccess;
11
12 public class Factory {
13
14     public static BLFacade sortuFactory(int mota) {
15         if(mota == 0) {
16             ConfigXML c=ConfigXML.getInstance();
17             DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
18             return new BLFacadeImplementation(da);
19         }
20         if(mota == 1) {
21             try {
22                 ConfigXML c=ConfigXML.getInstance();
23                 String serviceName= "http://"+c.getBusinessLogicNode()+"."+c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";
24                 URL url = new URL(serviceName);
25                 QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
26                 Service service = Service.create(url, qname);
27                 return service.getPort(BLFacade.class);
28             } catch (MalformedURLException e) {
29                 e.printStackTrace();
30             }
31         }
32         return null;
33     }
34 }

```

ApplicationLauncher-en baldintza *Factory* klasean egingo dugunez, kodea aldatu beharko dugu *Factory* klaseko metodora deitzeko eta implementazioa *Factory* klasean mugitu dugunez *ApplicationLauncher*-etik komentatuko dugu.

```

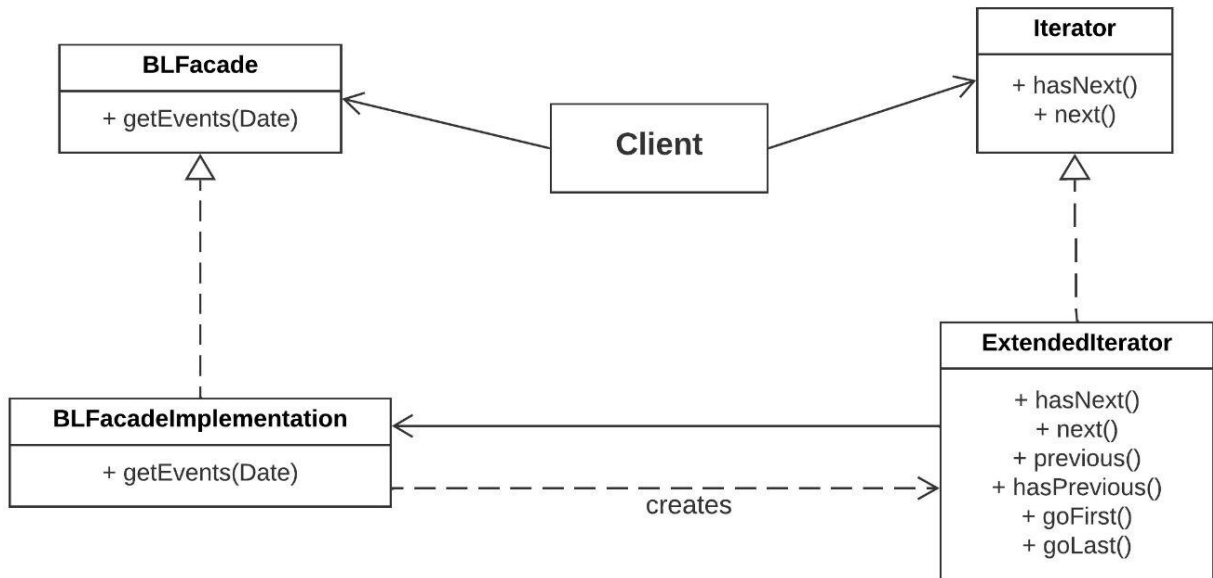
38     try {
39
40         BLFacade appFacadeInterface;
41         // UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel");
42         // UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
43         UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
44         if (c.isBusinessLogicLocal()) {
45
46             //In this option the DataAccess is created by FacadeImplementationWS
47             //appFacadeInterface=new BLFacadeImplementation();
48             //In this option, you can parameterize the DataAccess (e.g. a Mock DataAccess object)
49
50             //DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
51             //appFacadeInterface=new BLFacadeImplementation(da);
52             appFacadeInterface=Factory.sortuFactory(0);
53             System.out.println("Konexio lokala");
54         }
55         else { //If remote
56
57             // String serviceName= "http://"+c.getBusinessLogicNode()+"."+c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName();
58             //URL url = new URL("http://localhost:9999/ws/ruralHouses?wsdl");
59             //URL url = new URL(serviceName);
60             //1st argument refers to wsdl document above
61             //2nd argument is service name, refer to wsdl document above
62             //QName qname = new QName("http://businessLogic/", "FacadeImplementationWSService");
63             //QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
64             //Service service = Service.create(url, qname);
65             // appFacadeInterface = service.getPort(BLFacade.class);
66             appFacadeInterface=Factory.sortuFactory(1);
67             System.out.println("Urruneko konexioa");
68         }
69         //if (c.getDataBaseOpenMode().equals("initialize"))
70         appFacadeInterface.initializeRD();

```

Azkenik ikusi dezakegu zeinek betetzen duen *Creator*, *ConcreteProduct* eta *Product* rolak:

- **Creator** → *Factory* klaseak jokatzen du. Pasatako zenbakiaren arabera, negozio logikoa lokalean edo urrunean sortuko du.
- **ConcreteProduct** → *BLFacadeImplementation* klaseak jokatzen du, *BLFacade* interfazearen implementazioa jakin bat delako.
- **Product** → *BLFacade* jokatzen du, interfazea delako.

2. ITERATOR PATROIA



Lehenengo, **Iterator** klasea hedatzeko **ExtendedIterator** interfazea sortuko dugu. Domain paketea kokatu dugu iteratu beharreko objektua (**Event**) domain paketea dagoelako.

```
package domain;
```

```
import java.util.Iterator;
```

```
public interface ExtendedIterator<Event> extends Iterator<Event> {
    //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, **ExtendedIterator** interfazea inplementatuko duen **EventIterator** klasea sortuko dugu, domain paketea hau ere. Klase honetan eskatzen zitzaizkigun metodoak inplementatu ditugu eta bi atributu gehitu ditugu. Listaren posizioa jakiteko zenbaki bat eta **Event** motako objektuak izango dituen bektore bat.

```

package domain;

import java.util.Vector;

public class EventIterator<Event> implements ExtendedIterator<Event> {
    private Vector<Event> lista; //Eventu Lista
    private int posizioa;

    public EventIterator(Vector<Event> lista) {
        this.lista = lista;
        this.posizioa=0;
    }

    public Vector<Event> getEventList() {
        return this.lista;
    }

    @Override
    public boolean hasNext() {
        if(this.posizioa <= this.lista.size()-1) {
            return true;
        }else {
            return false;
        }
    }

    @Override
    public Event next() {
        Event e = lista.get(posizioa);
        this.posizioa++;
        return e;
    }

    @Override //?
    public Event previous() {
        Event e = lista.get(posizioa);
        posizioa--;
        return e;
    }

    @Override
    public boolean hasPrevious() {
        if(this.posizioa >= 0) {

```

```

        return true;
    }else {
        return false;
    }
}

@Override
public void goFirst() {
    this.posizioa=0;
}

@Override
public void goLast() {
    this.posizioa=this.lista.size()-1;
}

}

```

BLFacade eta **BLFacadeImplementation** klaseetan zegoen *getEvents()* metodoa aldatu dugu. Metodoak hasiera batean, Event motako objektuak gordetzen zituen bektore bat itzultzen zuen, aldaketen ostean ExtendedIterator<Event> bat itzultzen du.

BLFacade:

```
@WebMethod public ExtendedIterator<Event> getEvents(Date date);
```

BLFacadeImplementation lehen:

```
@WebMethod
public Vector<Event> getEvents(Date date) {
    dbManager.open(false);
    Vector<Event> events=dbManager.getEvents(date);
    dbManager.close();
    return events;
}

```

BLFacadeImplementation orain:

```
@WebMethod
public ExtendedIterator<Event> getEvents(Date date) {
    dbManager.open(false);
    Vector<Event> events=dbManager.getEvents(date);
    dbManager.close();
    return new EventIterator(events);
}

```

Aldaketak probatzeko **MainIterator** klasea sortu dugu. Klase honetan, **Factory** klaseari deituko diogu negozio logika sor dezan (lehenengo ariketan bezala), behin negozio logika sortuta, *getEvents()* metodoari deituko diogu **ExtendedIterator** motako iteradore bat lortuz, *i* deituriko iteradoreak ekintzak korrituko ditu, bai aurrerantz, eta bai atzerantz.

```
public class MainIterator {

    public static void main(String[] args) {

        boolean isLocal=true;
        //Facade objektua lortu lehendabiziko ariketa erabiliz
        BLFacade facadeInterface = Factory.sortuFactory(0);

        Calendar today = Calendar.getInstance();

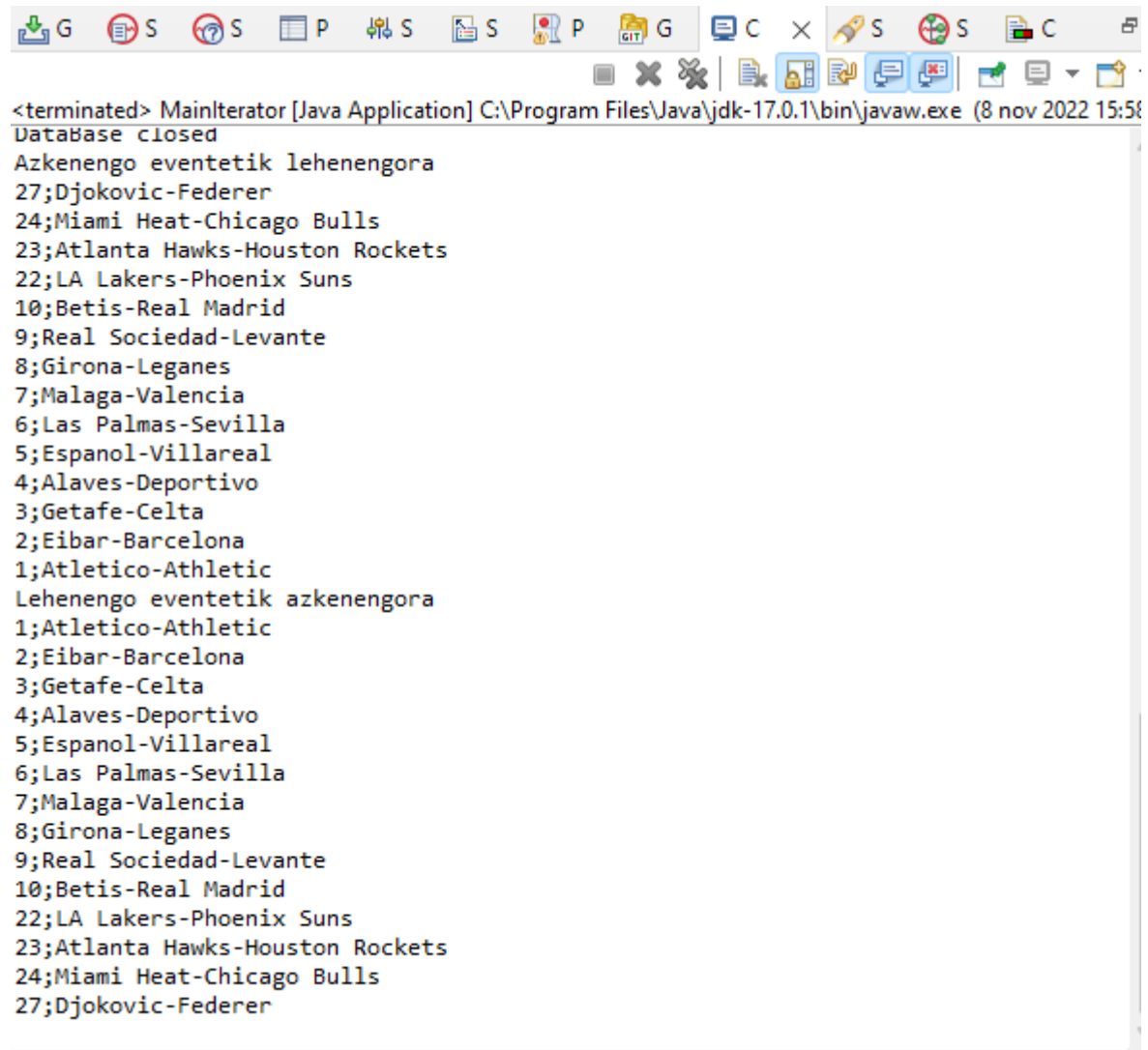
        int month=today.get(Calendar.MONTH);
        month+=1;
        int year=today.get(Calendar.YEAR);
        if (month==12) { month=0; year+=1;}

        ExtendedIterator<Event> i =
        facadeInterface.getEvents(UtilDate.newDate(year,month,17));
        Event ev;

        System.out.println("Azkenengo eventetik lehenengora");
        i.goLast();
        while (i.hasPrevious()){
            ev = (Event) i.previous();
            System.out.println(ev.toString());
        }

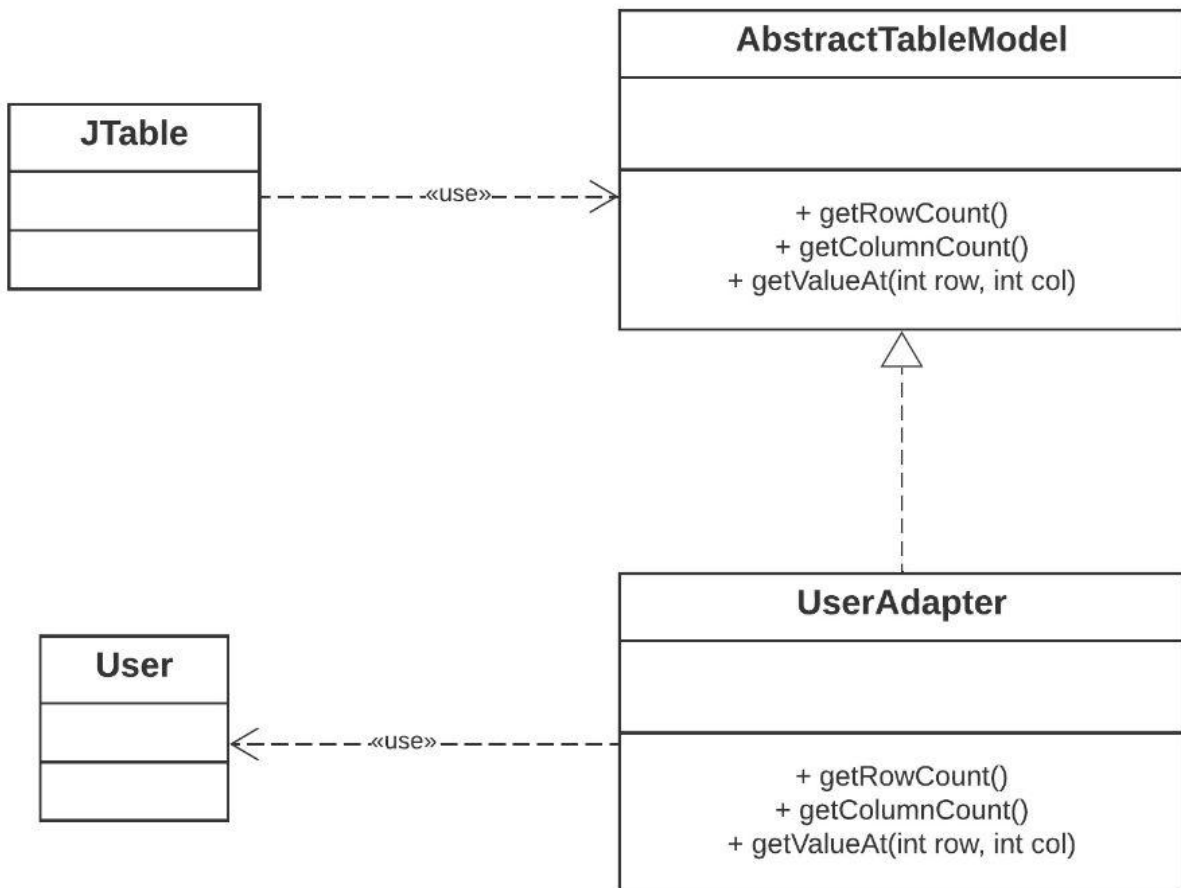
        System.out.println("Lehenengo eventetik azkenengora");
        //Nahiz eta suposatu hasierara ailegatu garela, eragiketa
        egiten dugu.
        i.goFirst();
        while (i.hasNext()){
            ev = i.next();
            System.out.println(ev.toString());
        }
    }
}
```


Exekutatu ondoren ondo egiten duela ikus dezakegu:



```
<terminated> Mainlterator [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (8 nov 2022 15:58)
DataBase closed
Azkenengo eventetik lehenengora
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
Lehenengo eventetik azkenengora
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
```

3. ADAPTER PATROIA



Ariketa honetan **JTable** batean erabiltzaile batek egin dituen apustu guztien informazioa aurkeztea eskatzen zaigu. Baina arazo bat dago, ezin dugu hori zuzenean implementatu **User** klaseak ez duelako **AbstractTableModel** implementatzen. Hori konpontzeko **AbstractTableModel** interfazea implementatzen duen **User** adaptadorea sortu dugu (diaposibetan zegoen [estekaz](#) baliatu gara implementazioa gauzatzeko).

```
public class UserAdapter extends AbstractTableModel{

    private String[] columnNames =
{"Event", "Question", "EventDate", "Bet(€)"};
    private Vector<ApustuAnitza> apustuak;

    public UserAdapter(Vector<ApustuAnitza> apustuak) {
        this.apustuak = apustuak;
    }
}
```

```

@Override
public int getRowCount() {
    int size;
    if (apustuak == null) {
        size = 0;
    }
    else {
        size = apustuak.size();
    }
    return size;
}

@Override
public int getColumnCount() {
    return columnNames.length;
}

@Override
public Object getValueAt(int rowIndex, int columnIndex) {
    ArrayList<Apustua> apu = new ArrayList<Apustua>();
    for(ApustuAnitza aa : apustuak) {
        for(Apustua a : aa.getApustuak()) {
            apu.add(a);
        }
    }

    if (columnIndex == 0) {
        return
apu.get(rowIndex).getKuota().getQuestion().getEvent().getDescription();
    }else if (columnIndex == 1) {
        return
apu.get(rowIndex).getKuota().getQuestion().getQuestion();
    }else if (columnIndex == 2) {
        return
apu.get(rowIndex).getKuota().getQuestion().getEvent().getEventDate();
    }
    return apu.get(rowIndex).getKuota().getQuote();
}

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

public Class getColumnClass(int col) {

```

```

        if (col == 3) {
            return Double.class;
        } else {
            return String.class;
        }
    }
}

```

Taula zuzen bistartzeko bost metodo horiek beharrezkoak dira. Kodearen azalpen pare bat. Lehen, erabiltzaile (**User**) batek ez dago zuzenean apustuekin lotuta, bai apustu anitzekin ordea, horregatik, *getValueAt()* metodoan apustu anitzek zituzten apustu guztiak **ArrayList<Apustua>** motako lista batean gorde ditugu, ondoren listako apustuak betetzen zuten baldintzaren arabera informazio konkretu bat itzuliko du.

Bigarrena, taulako zutabe bakoitza mota konkretu batekoa izango da eta beharrezkoa da zein izango den jartzea exekuzio garaian errorerik eman ez dezan, “*Cannot format given Object as a Number*” errorea hain zuzen. Hori ekiditeko apostatutako diru kopurua erakutsiko den zutabeak **Double** motako elementuak edukiko dituela konprobatu dugu, beste hirurak **String** motakoak direnez *else* batekin nahikoa da. Konprobaketa hau *getColumnClass()* metodoan egin dugu.

Erabiltzailearen apustuak erakutsiko dituen taula GUI berri batean jarri dugu, **EgindakoApustuakGUI** izenekoa. Hona hemen GUIaren inplementazioa:

```

public class EgindakoApustuakGUI extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTable taula;
    EgindakoApustuakGUI frame;

    public EgindakoApustuakGUI(Registered r) {
        this.frame = this;
        //frame.setVisible(true);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);

        UserAdapter use = new UserAdapter(r.getApustuAnitzak());
    }
}

```

```

        taula = new JTable(use);
        contentPane.add(taula, BorderLayout.CENTER);
        taula.setAutoCreateRowSorter(true);
        JScrollPane scrollPane = new JScrollPane(taula);
        scrollPane.setBounds(31, 5, 380, 280);
        scrollPane.setPreferredSize(new Dimension(380, 280));

        JPanel panel = new JPanel();
        panel.setLayout(null);
        panel.add(scrollPane);
        getContentPane().add(panel, BorderLayout.CENTER);
    }

}

```

GUI honetara joan ahal izateko **RegisteredGUI**-an botoi berri bat gehitu dugu.

```

jContentPane.add(getEgindakoApustuakButton());

private JButton getEgindakoApustuakButton() {
    if (btnEgindakoApustuak == null) {
        btnEgindakoApustuak = new JButton();
        btnEgindakoApustuak.setFont(new Font("Tahoma", Font.PLAIN,
16));
        btnEgindakoApustuak.setForeground(Color.DARK_GRAY);
        btnEgindakoApustuak.setBackground(Color.PINK);
        btnEgindakoApustuak.setOpaque(true);
        btnEgindakoApustuak.setBounds(63, 486, 181, 49);

        btnEgindakoApustuak.setText(ResourceBundle.getBundle("Etiquetas").getString("EgindakoApustuak"));
        btnEgindakoApustuak.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
e) {
                JFrame a = new EgindakoApustuakGUI((Registered)
user);
                a.setVisible(true);
            }
        }));
    }
    return btnEgindakoApustuak;
}

```

“markel” erabiltzaileak adibidez hiru apustu egin ditu.

Registered

Select Option

Event	Question	EventDate	Bet(€)
Real Madrid-B...	Emaizta?	Fri Oct 21 00:0...	2.5
Nadal-Alcaraz	Irabazlea?	Thu Dec 01 00:...	1.6
Djokovic-Feder...	Zeinek irabazik...	Sat Dec 17 00:...	2.3

Messages

Featured

Egindako apustuak

Close session