

SOFTWARE INGENIERITZA

2

PROIEKTUA: DISEINU PATROIAK

Egileak: Mikel Martin eta David Pintado

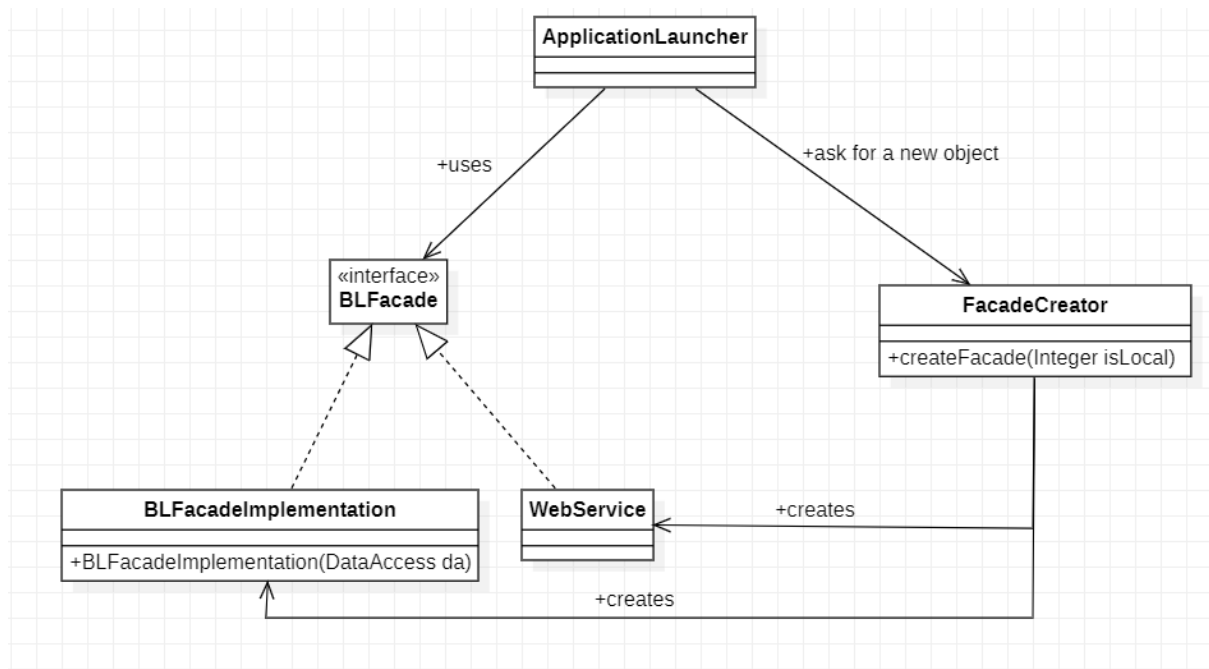
Data: 2022/11/12

GITHUB esteka: <https://github.com/mikel0912/BetCompleted22>

FACTORY METHOD PATROIA	3
1. UML HEDATUA	3
2. ALDATUTAKO KODEA	3
ITERATOR PATROIA	5
1. UML HEDATUA	5
2. ALDATUTAKO KODEA	6
3. EXEKUZIO IRUDIA	9
ADAPTER PATROIA	10
1. UML HEDATUA	10
2. ALDATUTAKO KODEA	10
3. EXEKUZIO IRUDIA	12

FACTORY METHOD PATROIA

1. UML HEDATUA



FacadeCreator klasea sortu dugu creator rola egingo duena, BLFacade interfazeak product rola egingo du eta interfaze hori inplementatzen duten bi klaseek, BLFacadeImplementation eta WebService, concreteProduct rola egingo dute.

BLFacadeImplementation: facade lokala

WebService: facade remotoa

2. ALDATUTAKO KODEA

Factory method patroia egin ahal izateko hainbat aldaketa egin ditugu gure kodean. Hasteko FacadeCreator klasea sortu dugu non createFacade metodoa daukagun. Metodo horri isLocal parametroa pasatzen zaio Integer motakoa. Parametro hori esaten digu BLFacade sortzeko egoera. Gure kasuan 1 parametroa pasata BLFacade lokala sortuko du eta 2 bada remotoa izango da.

CreateFacade metodo horretan honakoa ipini dugu:

```

public class FacadeCreator {
    public BLFacade createFacade(Integer isLocal) throws MalformedURLException {
        ConfigXML c=ConfigXML.getInstance();
        switch(isLocal) {
            case 1:
                DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
                return new BLFacadeImplementation(da);
            case 2:
                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);
            default:
                break;
        }
        return null;
    }
}

```

Lehenik eta behin ConfigXML fitxategiko instantzia c aldagaian hasieratzen dugu. Ondoren switch-case bat jartzen dugu, BLFacade nola sortzeko hainbat modu daudelako. Gure kasuan, lokala ala remotoa, baina gerta liteke etorkizunean beste sortzeko mota bat sartu izana eta horrela dagoen kodea ez da aldatu behar, soilik beste case bat gehitu.

Lokala izatekotan DataAccess berri bat sortzen dugu eta BLFacadeImplementation bat bueltatzen dugu DataAccess-a parametro bezala pasatuz.

Remotoa izatekoan, lehen bezala web zerbitzua dagoen tokiaren url-a sortze da eta zerbitzu horren portua bueltatzen dugu.

ApplicationLauncher klasean honakoa aldatu dugu:

```

FacadeCreator fc = new FacadeCreator();

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");
    Integer isLocal=1;

    appFacadeInterface = fc.createFacade(isLocal);

    MainGUI.setBussinessLogic(appFacadeInterface);

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

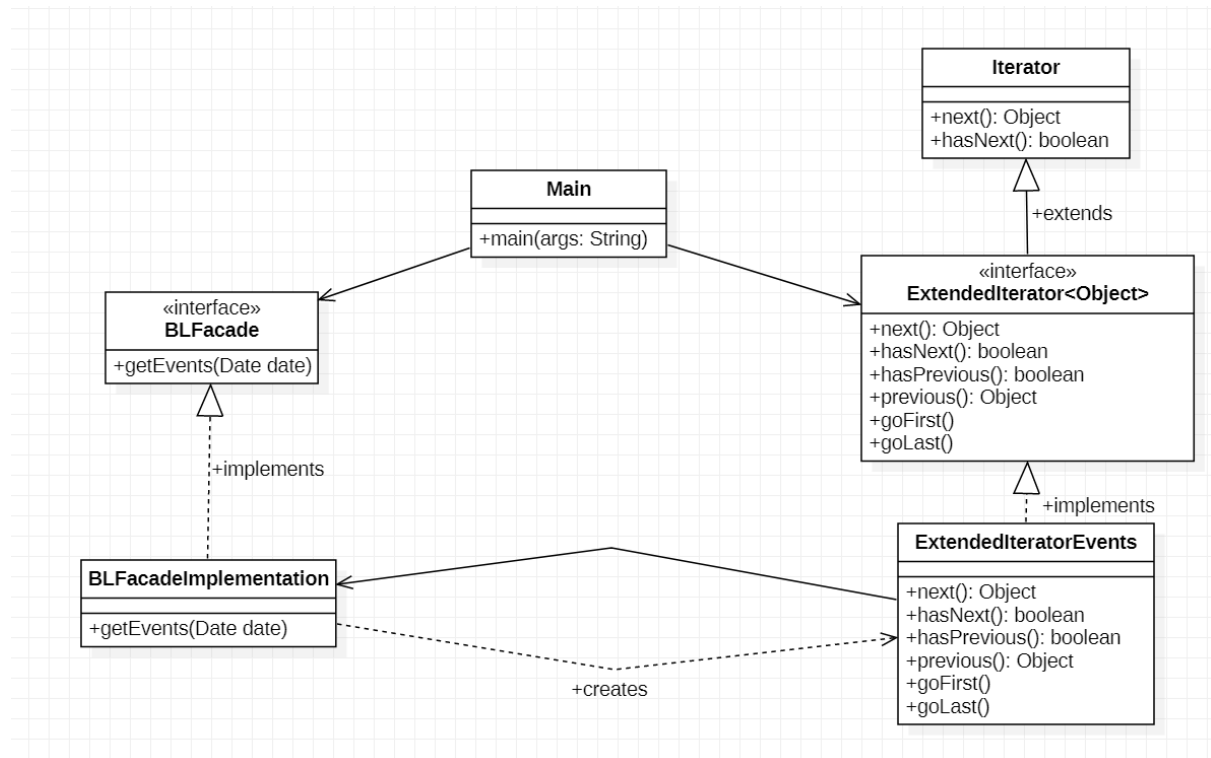
    System.out.println("Error in ApplicationLauncher: "+e.toString());
}

```

FacadeCreator klaseko aldagai berri bat sortu dugu, fc aldagaia. Lehen zegoen if baldintza kendu dugu eta fc-ri createFacade metodoa deitzen diogu. Parametro bezala guk hasieratutako Integer bat pasatzen diogu, horrela lehen aipatu dugun bezala facade lokala, remotoa edo beste berri bat nahiko bazen ere jakingo zen.

ITERATOR PATROIA

1. UML HEDATUA



UML-an agerikoa denez, Iterator kasetik eratortzen den `ExtendedIterator<Object>` interfazea sortu dugu, eta interfaze hori implementatzen duen `ExtendedIteratorEvents` klasea, biak Iterator patroia egikaritzeko sortutako `EventsIterator` paketea. Bi moduluak, iteradore bateko hainbat metodo dituzte.

Beste aldean, `BLFacade` interfazea daukagu (metodo guztiak ez jartzearren, asko direnez, soilik ITERATOR patroia egikaritzeko behar duguna jarri izan dugu: `getEvents(Date date)`), eta halaber, `BLFacadeImplementation` klasean berdina egin dugu. Klase honek `ExtendedIteratorEvents` erabiliko du, bat sortuz bere implementazioan.

Ondoren, erdian, client klasea dugu, `main` deituta, non bi aldeko klaseak erabiltzen dituen, probatzeko lehendabizi Event-ak alderantzizko ordenan korritzen direla, eta jarraian ohiko ordenan, sortu berri dugun Iterator patroiarekin.

Aipatzekoa da, beste metodo berri bat sortu dugula, `getEvents(Date date)` kopiaren bat, baina iteradorearekin zuzen aritzen dena. Izan ere, aurreikusi izan dugu, jada zegoena aldatzekotan, proiektuko leku askotan erroreak izango genituela, eta arazoak ez sortzeko hau egin dugu.

```
@WebMethod public ExtendedIterator<Event> getEvents2(Date date);
```

2. ALDATUTAKO KODEA

- **EXTENDEDITERATOR<OBJECT>**

Interfaze bat denez, soilik metodoen izenak agertuko dira:

```
public interface ExtendedIterator<Object> extends Iterator<Object>{
    public Object previous();
    public boolean hasPrevious();
    public void goFirst();
    public void goLast();
}
```

- **EXTENDEDITERATOREVENTS**

Klase honetan, aurreko metodoez implementatzeaz gain, beste bi izango ditu. Baina, zuzen aritzeko, lehendabizi iteratu nahi dugun lista bat sartuko dugu, baita posizio aldagai bat ere, lista eraikitzaileaz pasata:

```
public List<Event> ev;
public int position=0;
public ExtendedIteratorEvents(List<Event> ev) {
    this.ev=ev;
}
```

Ondoren metodoen inplementazioa izanez:

```
@Override
public boolean hasNext() {
    return position < ev.size();
}
```

```
@Override
public Event previous() {
    Event event =ev.get(position);
    position = position - 1;
    return event;
}
```

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

```
@Override
public Event next() {
    Event event =ev.get(position);
    position = position + 1;
    return event;
}
```

```
@Override
public boolean hasPrevious() {
    return position >= 0;
}
```

```
@Override
public void goLast() {
    position=ev.size()-1;
}
```

- **BLFACADE**

Lehen esan bezala, `getEvents2` sortu dugu, iterator-ekin aritzeko:

```
@WebMethod public ExtendedIterator<Event> getEvents2(Date date);
```

- **BLFACADEIMPLEMENTATION**

Aurreko metodoaren implementazioan itzultzen duena ze motakoa den aldatu dugu, `ExtendedIterator<Event>` bat itzultzeko:

```
@WebMethod
public ExtendedIterator<Event> getEvents2(Date date) {
    dbManager.open(false);
    ExtendedIterator<Event> events=dbManager.getEvents2(date);
    dbManager.close();
    return events;
}
```

- **DATAACCESS**

Lehen sortzen zen lista (gertaerak gordetzeko) erabiltzea `ExtendedIteratorEvents` bat sortzeko eta itzultzeko, lista hori parametroz pasata:

```
public ExtendedIterator<Event> getEvents2(Date date) {
    System.out.println(">> DataAccess: getEvents");
    List<Event> res = new ArrayList<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);
    }
    ExtendedIterator<Event> ema = new ExtendedIteratorEvents(res);
    return ema;
}
```

- **MAIN**

Main klase bat, non probatzen dugun ea sortu berri dugun patroia funtzionamendu egokia duen ala ez. Horretarako, `BLFacade` local bat sortu, eta `getEvents2`-rekin `ExtendedIterator<Event>` bat lortu, lista hori, iteradoreak dituen metodoekin korritzeko, atzetik aurrera, eta aurretik atzera ondoren:

```

public class Main {

    public static void main(String[] args) throws MalformedURLException {
        Integer isLocal=1;
        BLFacade blFacade = (new FacadeCreator()).createFacade(isLocal);
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Date date;
        try {
            date = sdf.parse("17/12/2022");
            ExtendedIterator<Event> i = blFacade.getEvents2(date);
            Event ev;
            System.out.println("_____");
            System.out.println("ATZETIK AURRERAKA");
            i.goLast();
            while (i.hasPrevious()) {
                ev = i.previous();
                System.out.println(ev.toString());
            }
            System.out.println();
            System.out.println("_____");
            System.out.println("AURRETIK ATZERAKA");
            i.goFirst();
            while (i.hasNext()) {
                ev = i.next();
                System.out.println(ev.toString());
            }
        } catch (ParseException e1) {
            System.out.println("Problems with date?? " +"17/12/2020");
        }

    }

}

```


3. EXEKUZIO IRUDIA

ATZETIK AURRERAKA

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-Leganés
 7;Málaga-Valencia
 6;Las Palmas-Sevilla
 5;Espanol-Villareal
 4;Alaves-Deportivo
 3;Getafe-Celta
 2;Eibar-Barcelona
 1;Atletico-Athletic

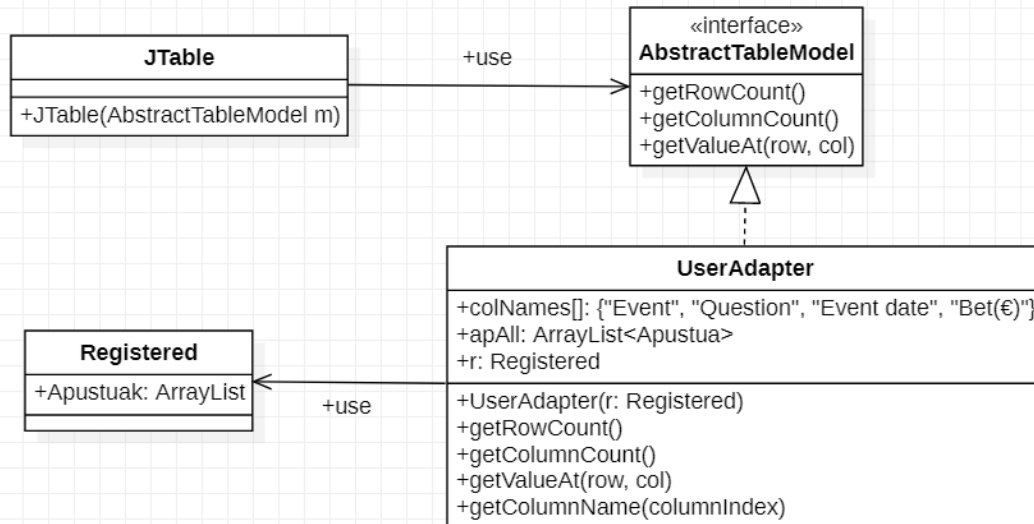
AURRETIK ATZERAKA

1;Atletico-Athletic
 2;Eibar-Barcelona
 3;Getafe-Celta
 4;Alaves-Deportivo
 5;Espanol-Villareal
 6;Las Palmas-Sevilla
 7;Málaga-Valencia
 8;Girona-Leganés
 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

Ondo ikusten denez irudian, lehendabizi iteradorea azken posizioan jartzen da, eta `getEvents()` metodoarekin lorturiko metodo guztiak atzetik aurrerako ordenean inprimatzen ditu; ondoren, hasierara itzuliz iteradorea, eta gertaera guztiak korrituz eta inprimatuz aurretik atzeraka.

ADAPTER PATROIA

1. UML HEDATUA



UMLan ikus daiteke adapter patroia jarraitu dugula **UserAdapter** klasea sortuz, klase hori **AbstractTableModel** interfazea eta **Registered** klasea erabiltzen ditu.

2. ALDATUTAKO KODEA

Adapter patroia jarraitzeko, lehendabizi Table Adapter paketea sortu izan dugu, eta bertan, **UserAdapter** deituriko klase bat sortu. Klase horrek, UML-an adierazi den bezala, Interfaze bat inplementatzen du, **AbstractTableModel** interfazea hain zuzen. Duen hiru metodoak inplementzeaz gain, **UserAdapter**-ek beste metodo bat izango du, `getColumnName()`, zutabe bat emanda bere zutabe izena itzultzeko.

Metodoak ondo funtzionatzeko, `colNames[]` deituriko aldagai bat izango dugu, zutabeen izenekin, erabiltzaile erregistratu bat (bere apustuak ikuskatu nahi dugulako), eta apustuen `ArrayList` bat, erabiltzaile erregistratuarenak hain zuzen:

```

public class UserAdapter extends AbstractTableModel{
    private String[] colNames = {"Event", "Question", "Event date", "Bet(€)"};
    private ArrayList<Apustua> apAll;
    private Registered r;
    public UserAdapter(Registered r) {
        this.r=r;
        apAll = new ArrayList<Apustua>();
        for(ApustuAnitza a: r.getApustuAnitzak()) {
            apAll.addAll(a.getApustuak());
        }
    }
}
  
```

Ondoren, metodoei dagokionez, batetik `getRowCount`(apustu arrayList luzera) eta `getColumnCount` (kasu honetan beti lau) ditugu:

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

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

Eta bestetik, `getColumnName`, zutabe indize bat emanda, zutabe horri dagokion izena itzultzeko, `colNames`-en begiratzuz:

```
public String getColumnName(int columnIndex) {
    return colNames[columnIndex];
}
```

Eta `getValueAt`, non zutabe eta errekada bat emanda, objektu bat ematen digun; izan ere, zutabearen arabera objektu motak ezberdinak baitira:

```
@Override
public Object getValueAt(int rowIndex, int columnIndex) {
    Object temp = null;
    if(columnIndex==0) {
        temp = apAll.get(rowIndex).getKuota().getQuestion().getEvent();
    }else if(columnIndex ==1) {
        temp = apAll.get(rowIndex).getKuota().getQuestion();
    }else if(columnIndex ==2) {
        temp= apAll.get(rowIndex).getKuota().getQuestion().getEvent().getEventDate();
    }else if(columnIndex ==3) {
        temp = apAll.get(rowIndex).getApustuAnitza().getBalioa();
    }
    return temp;
}
```

Ondoren, emaitza ikustatzeko eta egiaztatzeko, `WindowTableGUI` bat sortu dugu. Baina lehendabizi, hori ikustatzeko, `RegisteredGUI`-an kodea gehitu izan dugu, `WindowTableGUI`-ra joateko, jarri dugu botoi berrian klikatuz gero (ikusi hurrengo ataleko exekuzio irudia). Horretarako, `RegisteredGUI`-n:

```
JButton btnNewButton_1 = new JButton("Table of "+user); //$NON-NLS-1$ //$NON-NLS-2$
    btnNewButton_1.setFont(new Font("Tahoma", Font.PLAIN, 16));
    btnNewButton_1.setForeground(Color.DARK_GRAY);
    btnNewButton_1.setBackground(Color.PINK);
    btnNewButton_1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JFrame a =new WindowTableGUI((Registered)user);
            a.setVisible(true);
        }
    });
    btnNewButton_1.setBounds(10, 500, 282, 53);
    jContentPane.add(btnNewButton_1);
```

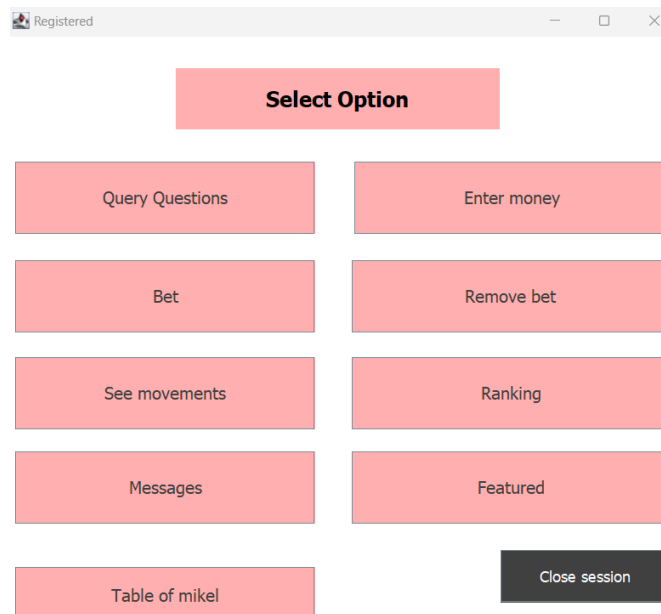
Eta azkenik, WindowTableGUI klasea dugu, zein JFrame heredatzen duen. Klase honetan, Jtable, UserAdapter eta Registered objektuak izango ditugu aldagia bezala, eta eraikitzailean erazagutu, eta table instantziatu UserAdapterarekin, ondoren table-a ondo jartzeko behar diren aginduak jarritz:

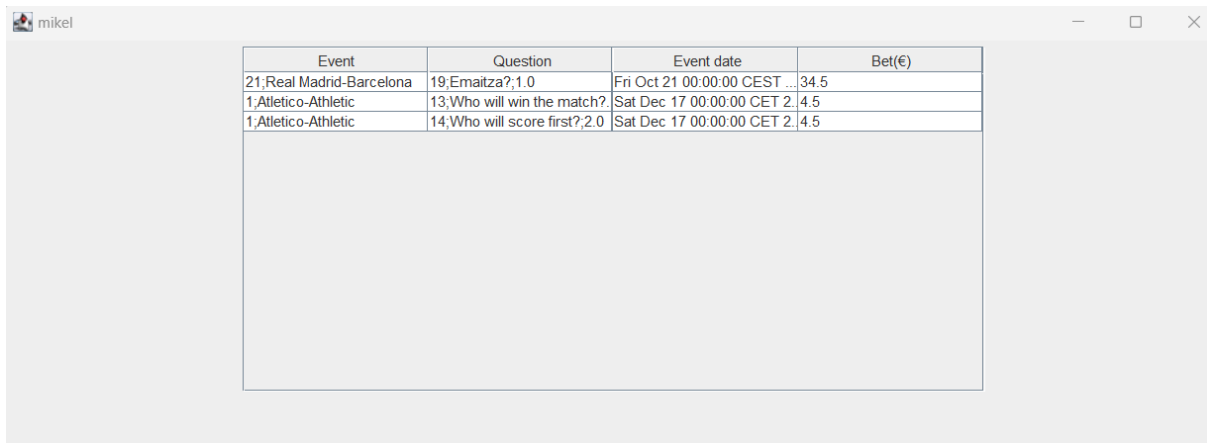
```
public class WindowTableGUI extends JFrame{
    private JTable table;
    private UserAdapter ua;
    private Registered u;
    public WindowTableGUI(Registered user) {
        super(user.getUsername());
        setBounds(10,10,1000,600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.u=user;

        ua= new UserAdapter(u);

        table = new JTable(ua);
        table.setBounds(85, 49, 243, 121);
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setPreferredSize(new Dimension(600,280));
        JPanel panel = new JPanel();
        panel.add(scrollPane);
        add(panel,BorderLayout.CENTER);
    }
}
```

3. EXEKUZIO IRUDIA





Event	Question	Event date	Bet(€)
21;Real Madrid-Barcelona	19;Eraitza?;1.0	Fri Oct 21 00:00:00 CEST ...	34.5
1;Atletico-Athletic	13;Who will win the match?	Sat Dec 17 00:00:00 CET 2.	4.5
1;Atletico-Athletic	14;Who will score first?;2.0	Sat Dec 17 00:00:00 CET 2.	4.5

Ikus daitekeen bezala behean ezkerrean txertatu dugu RegisteredGULan botoi berri bat. Botoi horri emanez 2. irudia agertuko da saioa hasi duen erabiltzailearen apustuen informazioarekin.