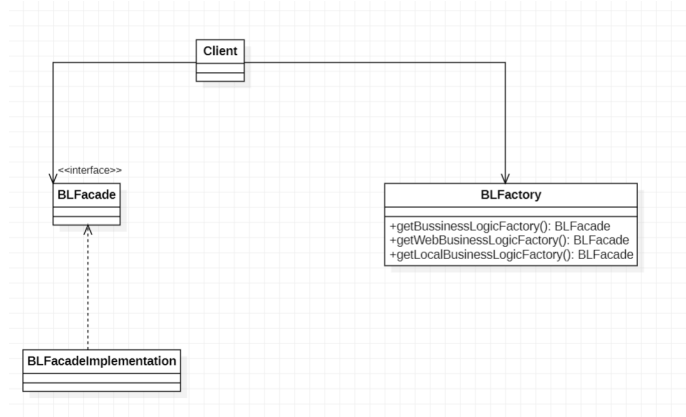


Patrones de diseño:

Patrón Factory Method:

Diagrama UML:



Código modificado:

Para realizar el patrón factory, hemos añadido en el package Factory una clase **BLFactory** el cual hará el papel de **Creator**, **BLFacade** realizará el papel de **Product** y por último **BLFacadeImplementation** será quien haga el papel de **ConcreteProduct**.

Clase BLFactory:

```
package factory;

import java.net.MalformedURLException;

public class BLFactory {

    ConfigXML c = ConfigXML.getInstance();

    public void BLFactory(){}

    public BLFacade getBusinessLogicFactory() throws MalformedURLException {
        BLFacade appFacadeInterface ;

        if(c.isBusinessLogicLocal()) {
            appFacadeInterface= getLocalBusinessLogicFactory();
        }
        else {
            appFacadeInterface= getWebBusinessLogicFactory();
        }
        return appFacadeInterface;
    }

    private BLFacade getWebBusinessLogicFactory() throws MalformedURLException {
        BLFacade appFacadeInterface ;
        String serviceName= "http://"+c.getBusinessLogicNode()+"-"+c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";

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

```

//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);

appFacadeInterface = service.getPort(BLFacade.class);

return appFacadeInterface;

}

private BLFacade getLocalBusinessLogicFactory() {
    BLFacade appFacadeInterface ;

    DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));

    appFacadeInterface =new BLFacadeImplementation(da);

    return appFacadeInterface;
}
}

```

En esta clase, hemos implementado tres métodos:

- **getBuissinessLogicFactory():** Es el método que se llama desde la clase ApplicationLauncher, y es el encargado de decidir si la BLFacade que se va a crear es local o web
- **getWebBuissinessLogicFactory():** Este método se llama desde el método anteriormente descrito, y su función es generar la BLFacade de manera local.
- **getLocalBuissinessLogicFactory():** Este método realiza la misma tarea que el anterior, sin embargo, este genera la BLFacade de manera web.

Clase ApplicationLauncher:

```

public static IniciarSesionGUI inicioSesion() {
    return inicioSesion;
}

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

    System.out.println(c.getLocale());

    inicioSesion = new IniciarSesionGUI();

    Locale.setDefault(new Locale(c.getLocale()));

    System.out.println("Locale: "+Locale.getDefault());

    inicioSesion.setVisible(true);

    try {
        BLFacade appFacadeInterface= new BLFactory().getBusinessLogicFactory();
        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");

        /*if (c.getDataBaseOpenMode().equals("initialize"))
            appFacadeInterface.initializeBD();
        */

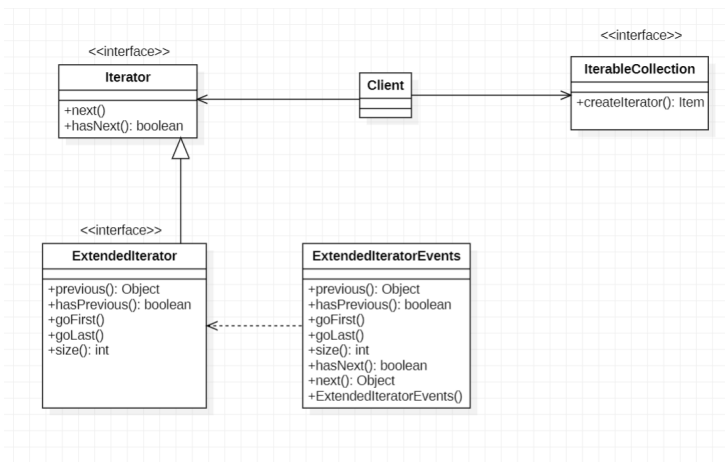
        IniciarSesionGUI.setBuissinessLogic(appFacadeInterface);
    }
}

```

En la clase ApplicationLauncher, el cambio ha sido significativo, ya que ahora en vez de realizar un if/else para cada respectivo caso (Local/Web), se realiza una única llamada a BLFactory, que es la encargada de decidir si será local o web.

Patrón Iterator:

Diagrama UML:



Código modificado:

Para realizar este patrón, hemos añadido un nuevo package “ExtendedIterator”, en el cual hemos añadido la interfaz `ExtendedIterator.java` y `ExtendedIteratorEvents.java`.

ExtendedIterator.java:

La interfaz tendrá los métodos necesarios para la clase `ExtendedIteratorEvents`.

```
1 package extendedIterator;
2
3 import java.util.Iterator;
4
5
6
7
8 public interface ExtendedIterator<Object> extends Iterator<Object> {
9
10     //return the actual element and go to the previous
11
12     public Object previous();
13     //true if there is a previous element
14
15     public boolean hasPrevious();
16     //It is placed in the first element
17
18     public void goFirst();
19     //It is placed in the last element
20
21     public void goLast();
22
23     public int size();
24 }
```

ExtendedIteratorEvents.java:

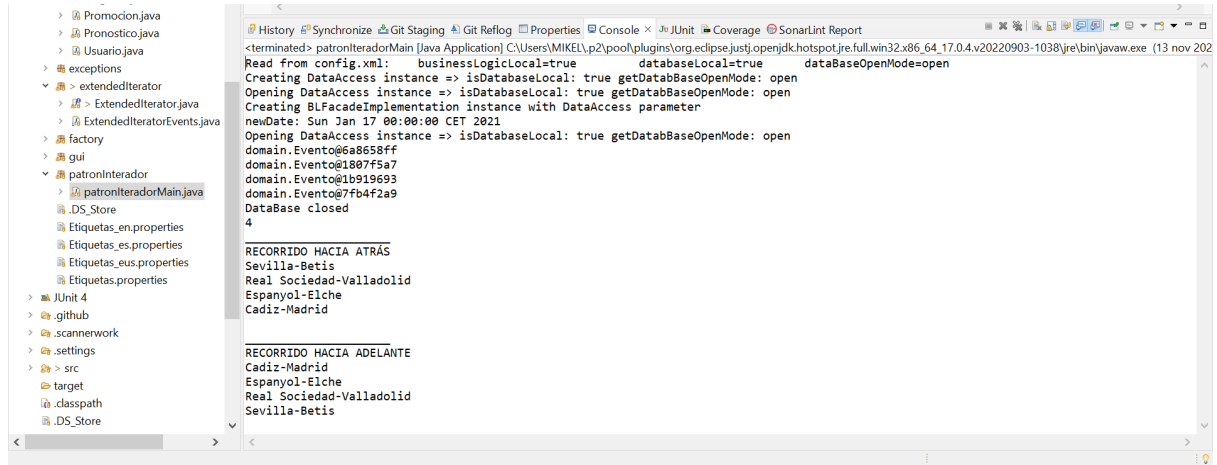
La clase implementa los métodos de la interfaz para poder recorrer los eventos. previous(), hasPrevious(), goLast(), goFirst()...

Esta clase tendrá dos atributos: un vector de eventos, y la posición actual de dicho vector, en la constructora la posición se inicializará a 0 y el vector se inicializará con el vector de eventos obtenidos desde el método getEvents(date).

```
1 package extendedIterator;
2
3 import java.util.Vector;
4
5
6 public class ExtendedIteratorEvents implements ExtendedIterator<Evento>{
7
8     private Vector<Evento> eventos;
9     private int indice=0;
10
11     public ExtendedIteratorEvents(Vector<Evento> e) {
12         this.eventos=e;
13     }
14
15     @Override
16     public int size() {
17         return eventos.size();
18     }
19
20     @Override
21     public boolean hasNext() {
22         return indice<eventos.size();
23     }
24
25     @Override
26     public Evento next() {
27         Evento ev=eventos.get(indice);
28         indice++;
29         return ev;
30     }
31
32     @Override
33     public Evento previous() {
34         Evento ev=eventos.get(indice);
35         indice--;
36         return ev;
37     }
38
39     @Override
40     public boolean hasPrevious() {
41         return indice>0;
42     }
43
44     @Override
45     public void goFirst() {
46         indice=0;
47     }
48
49     @Override
50     public void goLast() {
51         indice=eventos.size()-1;
52     }
53 }
```

Captura imagen:

Resultado obtenido en consola tras ejecutar el código main proporcionado.



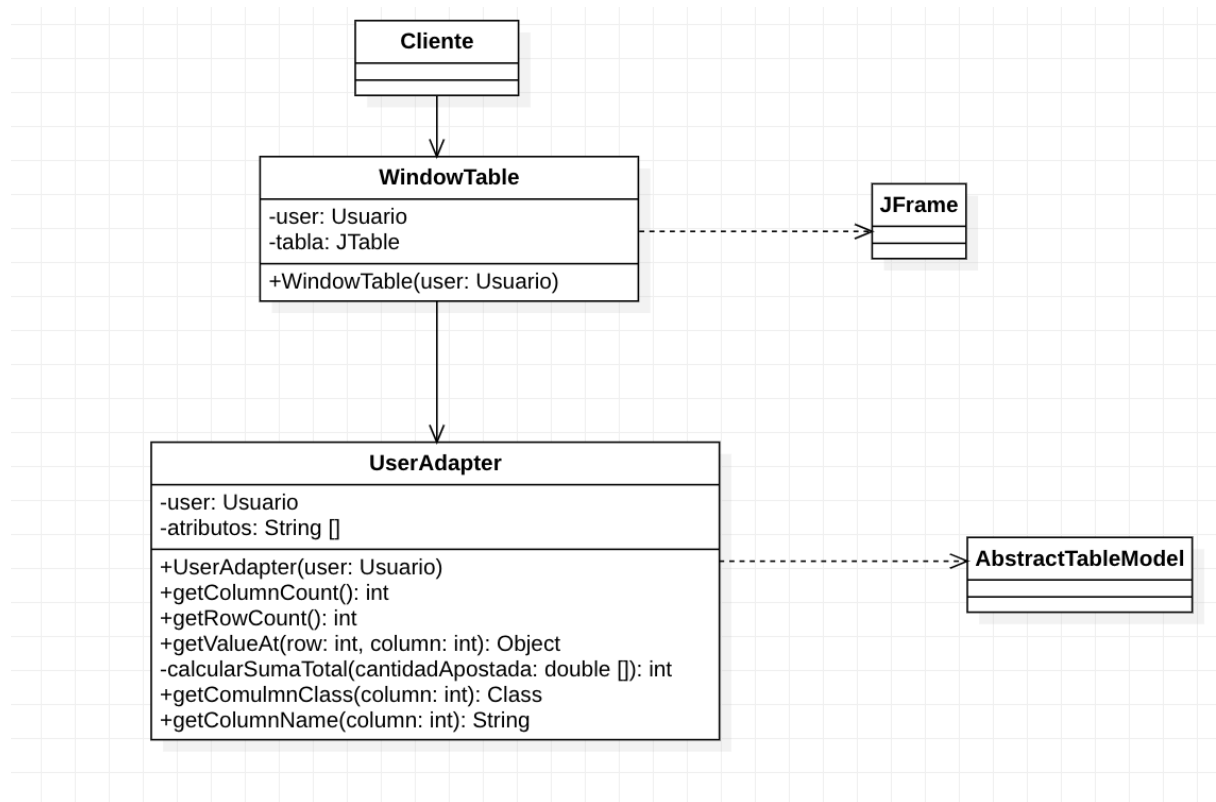
```
<terminated> patronIteradorMain [Java Application] C:\Users\MIKEL\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe (13 nov 2022)
Read from config.xml:  businessLogicLocal=true      databaseLocal=true      dataBaseOpenMode=open
Creating DataAccess instance => isDatabaseLocal: true getDatabaseOpenMode: open
Opening DataAccess instance => isDatabaseLocal: true getDatabaseOpenMode: open
Creating BLFacadeImplementation instance with DataAccess parameter
newDate: Sun Jan 17 00:00:00 CET 2021
Opening DataAccess instance => isDatabaseLocal: true getDatabaseOpenMode: open
domain.Evento@6a8658ff
domain.Evento@1807f5a7
domain.Evento@1b919693
domain.Evento@7fb4f2a9
DataBase closed
4

RECORRIDO HACIA ATRÁS
Sevilla-Betis
Real Sociedad-Valladolid
Espanyol-Elche
Cadiz-Madrid

RECORRIDO HACIA ADELANTE
Cadiz-Madrid
Espanyol-Elche
Real Sociedad-Valladolid
Sevilla-Betis
```

Patrón Adapter:

Diagrama UML:



Código modificado:

Clase WindowTable.java:

```
package gui;

import java.awt.Dimension;

public class WindowTable extends JFrame{

    private Usuario user;
    private JTable tabla=new JTable();

    public WindowTable(Usuario user){

        super("Apuestas realizadas por "+ user.getNombreUsuario()+":");
        this.setBounds(100, 100, 700, 200);
        this.user = user;
        UserAdapter adapt = new UserAdapter(user);
        tabla = new JTable(adapt);
        tabla.setAutoCreateRowSorter(true);
        tabla.setPreferredScrollableViewportSize(new Dimension(500, 70)); //C
        JScrollPane scrollPane = new JScrollPane(tabla); //Agregamos el JScro
        getContentPane().add(scrollPane, BorderLayout.CENTER);

    }
}
```

En esta clase no ha habido ninguna modificación.

Clase UserAdapter.java:

```
1 package userAdapter;
2
3 import java.util.Date;
4
5
6
7
8
9
10
11
12
13 public class UserAdapter extends AbstractTableModel {
14
15
16     private Usuario user;
17
18     private String [] atributos= {"Pronostico", "Pregunta", "Fecha Primera Apuesta", "Cantidad Total Apostada"};
19
20
21
22
23     public UserAdapter(Usuario user) {
24         this.user=user;
25     }
26
27     public int getColumnCount() {
28         return atributos.length;
29     }
30
31
32
33     public int getRowCount() {
34         int size;
35
36         if(user.getApuestas()==null) {
37             size=0;
38         }
39
40         else {
41             size=user.getApuestas().size();
42         }
43         return size;
44     }
45
46     public Object getValueAt(int row, int col) {
47         if(col==0) {
48             return user.getApuestas().get(row).getPronostico().getRespuesta();
49         }
50         if(col==1) {
51             return user.getApuestas().get(row).getPronostico().getPregunta().getEnunciado() + " En el partido: " +
52                 user.getApuestas().get(row).getPronostico().getPregunta().getEvento().getNombre();
53         }
54         if(col==2) {
55             return user.getApuestas().get(row).getFecha()[0];
56         }
57
58         if(col==3) {
59             return calcularSumaTotal(user.getApuestas().get(row).getCantidadApostada());
60         }
61         return null;
62     }
63
64
65     private Object calcularSumaTotal(double[] cantidadApostada) {
66         double suma=0;
67         for(Double d: cantidadApostada) {
68             suma+=d;
69         }
70         return suma;
71     }
72
73     public String getColumnName(int col) {
74         return atributos[col];
75     }
76
77     public Class getColumnClass(int col) {
78         return String.class;
79     }
80
81
82
83
84 }
85
```

En esta clase, hemos hecho varias modificaciones:

- Hemos añadido el atributo *atributos*, para almacenar el nombre de las columnas, y así hacer el método `getColumnName()` más legible.
- El método de `getValueAt(int, int)`, lo hemos implementado de tal manera que según la fila en la que sea, accede a la apuesta que le corresponda, y según la columna, accede al atributo correspondiente de dicha apuesta.

- Como se puede observar, hemos añadido también un método llamado `calcularSumaTotal(...)`. Este método es necesario ya que de la manera en la que tenemos implementadas las apuestas, la cantidad apostada se guarda en un array, por si el usuario hace mas de una apuesta, tener constancia de lo que apuesta en cada una. Por tanto, para poder visualizar la cantidad total, es necesario dicho método.

Captura imagen:

Captura de la ejecución del main proporcionado:

Apuestas realizadas por Usuario1:			
Pronostico	Pregunta	Fecha Primera Apuesta	Cantidad Total Apostada
No	¿Habrán goles en la primera parte? En el partido: Atlético-Athletic	Sun Nov 13 19:02:14 CET 2022	100.0
Unai Simon a favor.	¿Quién meterá el primer gol? En el partido: Atlético-Athletic	Sun Nov 13 20:22:47 CET 2022	45.0
Elbar	¿Quién ganará el partido? En el partido: Elbar-Barcelona	Sun Nov 13 20:22:26 CET 2022	34.0