Ingeniería del Software II

Patrones de Diseño

Informe

Alexandra Aleina Pelipian
Ignacio Ayaso Hernández
Josu Lorenzo Hernández

Facultad de Informática
Grado de Ingeniería Informática

30 de Octubre, 2022

Índice

Integrantes del Equipo

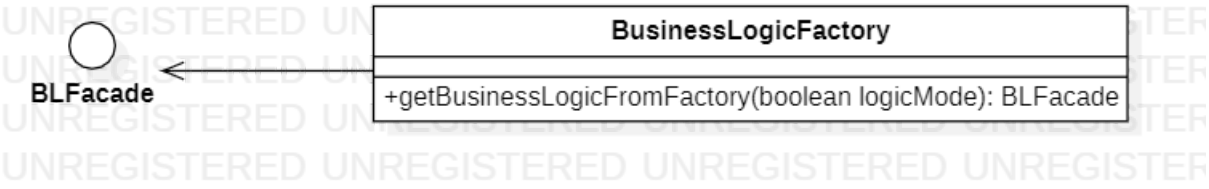
Los integrantes del equipo desarrollador son los siguientes:

Alexandra Aleina Pelipian
Ignacio Ayaso Hernández
Josu Lorenzo Hernández

Apartado 1. Patrón Factory Method

Autor: Josu Lorenzo Hernández

Dedicación: 3 horas



Partes significativas del código: creación y uso de la factoría en la clase ApplicationLauncher.js.

```
BusinessLogicFactory blfactory = new BusinessLogicFactory();
boolean blmode = false;

BLFacade appFacadeInterface = blfactory.getBusinessLogicFromFactory(blmode);

UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

MainGUI.setBussinessLogic(appFacadeInterface);
```

Partes significativas del código: implementación de la factoría en la clase BusinessLogicFactory.js

```
public class BusinessLogicFactory {
    public BLFacade getBusinessLogicFromFactory(boolean logicMode) {

        BLFacade appFacadeInterface = null;

        if (logicMode) {
            // server
            ConfigXML c=ConfigXML.getInstance();

            String serviceName= "http://"+c.getBusinessLogicNode() +":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";

            try {
                URL url;
                url = new URL(serviceName);
                QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
                Service service = Service.create(url, qname);

                appFacadeInterface = service.getPort(BLFacade.class);

            } catch (MalformedURLException e) {
                e.printStackTrace();
                System.out.println("Error obtaining remote businessLogic");
            }

        } else {
            // local businessLogic

            appFacadeInterface = new BLFacadeImplementation(false);
        }

        return appFacadeInterface;
```

Conclusiones

El patrón de diseño “Factory” se fundamenta en crear una entidad instanciadora. Esta entidad está pensada para ser ampliada de cara a futuro, y a la vez ser fácil de mantener y mejorar a lo largo del ciclo de vida del software. La factoría está pensada como una clase superior, la cual tiene las responsabilidades de crear y devolver las instancias de objetos clave en un programa; en nuestro caso, la lógica de negocio. Además, debido a que está pensada para ser fácilmente ampliable y devolver diferentes resultados ya configurados, la factoría está parametrizada. En pocas palabras, es un objeto instanciador, gracias al cual la legibilidad del código aumenta considerablemente.

Apartado 2. Patrón Iterator

Autor: Ignacio Ayaso
Dedicación: 3 hrs

En diseño de software, el patrón de diseño Iterator, define una interfaz que declara los métodos necesarios para acceder secuencialmente a un grupo de objetos de una colección.

En este caso se va un poco más allá, queremos darle la soltura para que se pueda recorrer tanto de derecha a izquierda como de izquierda a derecha, para ello se han creado e implementado los siguientes métodos:

next(): Este método lo usamos para poder ir al siguiente elemento de objeto Iterator;

previous(): Este método lo usamos para poder ir al elemento anterior contenido en el objeto Iterator;

hasNext(): Se usa para saber si, en la posición actual en la que nos encontramos recorriendo la lista hay un elemento siguiente;

hasPrevious(): Se usa para saber si, en la posición actual en la que nos encontramos recorriendo la lista hay un elemento siguiente;

goFirst(): Este método se usa en caso de querer que el iterador se posicione en el primer elemento del objeto iterable;

goLast(): Este método se usa en caso de querer que el iterador se posicione en el último elemento del objeto iterable;

Este patrón de diseño permite recorrer una estructura de datos sin que sea necesario conocer la estructura interna de la misma.

En consecuencia, he obtenido el siguiente código en el paquete de Dominio:

```java
package domain;

import java.util.Iterator;

public interface ExtendedIterator<T> extends Iterator<Event>{

    //return the actual element and go to the previous
    public T previous();

    //true if there is a previous element
    public boolean hasPrevious();

    //It is placed in the first element
    public void goFirst();

    //It is placed in the last element
    public void goLast();
}

```

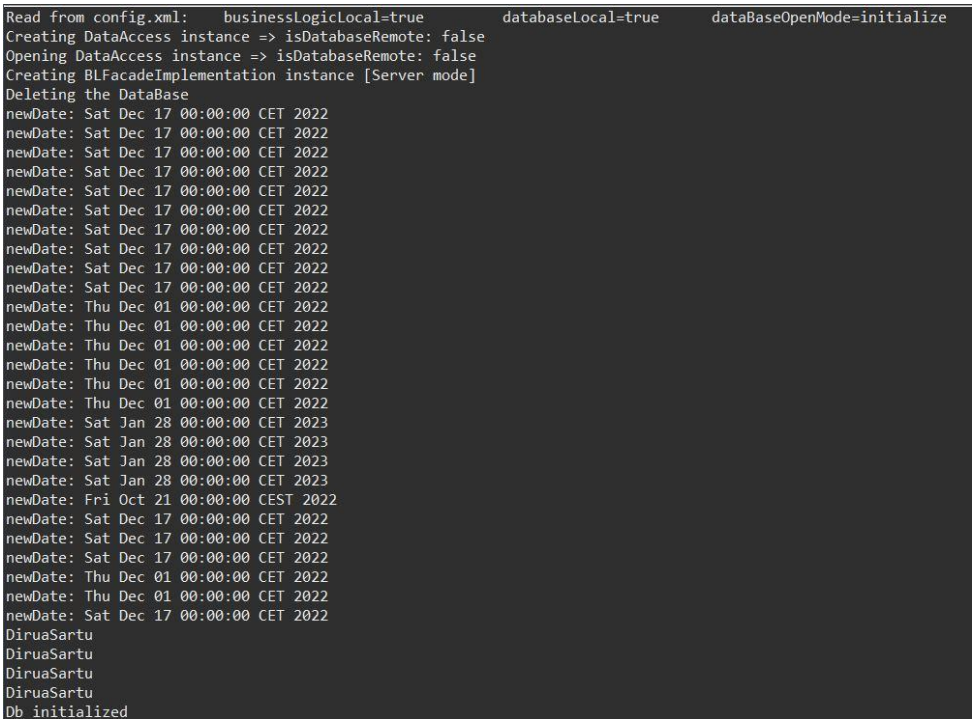
```java
package domain;

import java.util.Vector;

public class ExtendedIteratorEvents implements ExtendedIterator<Event>{

    private Vector<Event> vec;
    private Event actual;
    private int indexOfActual;


    public ExtendedIteratorEvents(Vector<Event> vec) {
        this.vec = vec;
        this.actual = vec.firstElement();
        this.indexOfActual = vec.indexOf(vec.firstElement());
    }

    public Event next() {
        actual = vec.get(indexOfActual+1);
        indexOfActual = vec.indexOf(actual);
        return actual;
    }

    public Event previous() {
        this.actual = vec.get( indexOfActual-1);
        this.indexOfActual = this.vec.indexOf(actual);
        return actual;
    }

    public boolean hasNext() {
        if(this.indexOfActual < this.vec.size()-1) {
            return true;
        }else {
            return false;
        }
    }

    public boolean hasPrevious() {
        if(this.indexOfActual > 0){
            return true;
        }else {
            return false;
        }
    }

    public void goFirst() {
        this.actual = vec.get(0);
        this.indexOfActual= vec.indexOf(actual)-1;
    }

    public void goLast() {
        this.actual = vec.get(vec.size()-1);
        this.indexOfActual = vec.indexOf(actual)+1;

    }

}

```

y este es el caso de prueba:

```java
package domain;

import java.text.ParseException;

public class MainExtendedIterator {
    public static void main(String[] args)  {
        //obtener el objeto Facade local
        boolean isLocal = false;
        BLFacade blFacade = new BusinessLogicFactory().getBusinessLogicFromFactory(isLocal);
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Date date;
        try {
            date = sdf.parse("17/12/2022"); // 17 del mes que viene
            ExtendedIterator<Event> i = blFacade.getEventsIterator(date);
            Event e;
            System.out.println("_____________________");
            System.out.println("RECORRIDO HACIA ATRÁS");
            i.goLast(); //  Hacia atrás
            while (i.hasPrevious()) {
                e = i.previous();
                System.out.println(e.toString());
            }
            System.out.println();
            System.out.println("_____________________");
            System.out.println("RECORRIDO HACIA ADELANTE");
            i.goFirst();    //  Hacia adelante
            while (i.hasNext()) {
                e = i.next();
                System.out.println(e.toString());
            }
        }   catch (ParseException e1)   {
            System.out.println("Problems with date?? " + "17/12/2020");
        }
        System.out.println();
    }
}
```

Tal y como se indica se tiene que mostrar lo que devuelve este “main”:

(La tercera imagen muestra el correcto funcionamiento de tanto el método getEventsIterator(Date date) como de los métodos implementados en la clase ExtendedIteratorEvents)

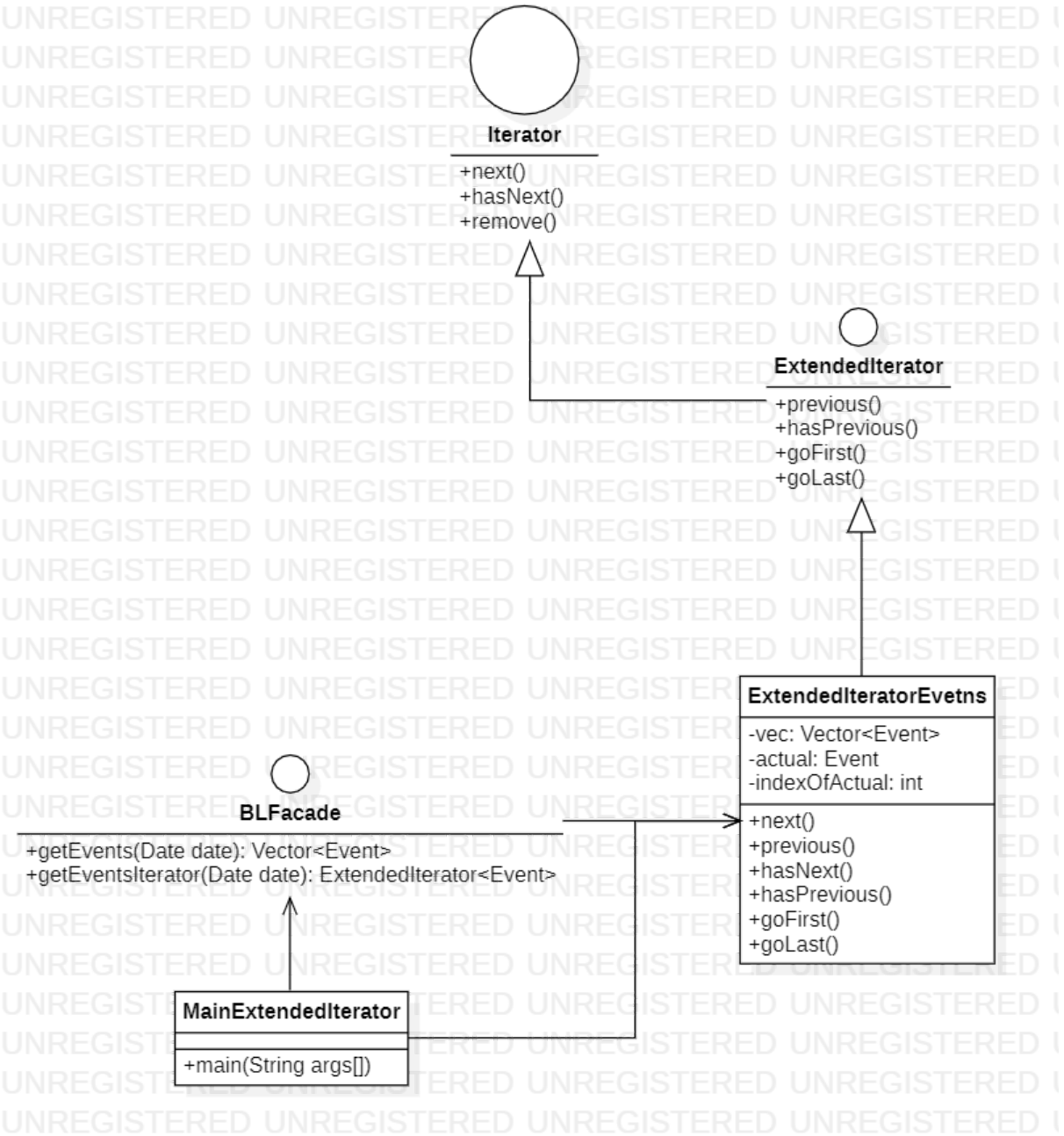


```
Read from config.xml:    businessLogicLocal=true        databaseLocal=true      dataBaseOpenMode=initialize
Creating DataAccess instance => isDatabaseRemote: false
Opening DataAccess instance => isDatabaseRemote: false
Creating BLFacadeImplementation instance [Server mode]
Deleting the DataBase
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Sat Jan 28 00:00:00 CET 2023
newDate: Sat Jan 28 00:00:00 CET 2023
newDate: Sat Jan 28 00:00:00 CET 2023
newDate: Sat Jan 28 00:00:00 CET 2023
newDate: Fri Oct 21 00:00:00 CEST 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Thu Dec 01 00:00:00 CET 2022
newDate: Sat Dec 17 00:00:00 CET 2022
DiruaSartu
DiruaSartu
DiruaSartu
DiruaSartu
Db initialized
```

```
Db initialized
DataBase closed
Opening DataAccess instance => isDatabaseRemote: false
Creating BLFacadeImplementation instance [Server mode]
Deleting the DataBase
>> DataAccess: getEvents
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
DataBase closed
```

```
_____________________
RECORRIDO HACIA ATRÁS
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


_____________________
RECORRIDO HACIA ADELANTE
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
```

Diagrama UML :


Iterator
+next()
+hasNext()
+remove()
ExtendedIterator
+previous()
+hasPrevious()
+goFirst()
+goLast()
ExtendedIteratorEvetns
-vec: Vector<Event>
-actual: Event
-indexOfActual: int
+next()
+previous()
+hasNext()
+hasPrevious()
+goFirst()
+goLast()
BLFacade
+getEvents(Date date): Vector<Event>
+getEventsIterator(Date date): ExtendedIterator<Event>
MainExtendedIterator
+main(String args[])

Apartado 3. Patrón Adapter

Autor: Alexandra Aleina Pelipian
Dedicación: 2 horas
Notas:

1. Para realizar un adapter adecuado de la clase Registered, con todas sus apuestas, necesitaría un atributo propio que contenga la lista de apuestas, y no de las apuestas combinadas. En el caso de este proyecto, debería de modificar la clase Registered, caso que no me es permitido.
2. Me he encontrado con la dificultad de entender el código en euskera, ya que no hablo el idioma y me ha costado más de lo esperado traducir **y** entender varios métodos a lo largo del desarrollo de las tareas.

En consecuencia, he obtenido el siguiente código en el paquete de Dominio:

```
package domain;

import java.util.List;

public class UserAdapter extends AbstractTableModel {
    private final Vector<ApustuAnitza> apuestas;
    private Registered user;
    private String[] colNames = new String[] {"Event", "Question", "Event Date", "Bet (€)"};
    public UserAdapter(Registered user) {
        apuestas = new Vector<ApustuAnitza>(user.getApustuAnitzak());
        this.user = user;
    }
    public int getRowCount() {
        return apuestas.size();
    }
    public int getColumnCount() {
        return 4;
    }
    public String getColumnName(int col) {
        return colNames[col];
    }
    public Object getValueAt(int rowIndex, int columnIndex) {
        switch(columnIndex) {
        case 0: return (Object)apuestas.get(rowIndex).getApustuak();
        case 1: return (Object)apuestas.get(rowIndex);
        case 2: return (Object)apuestas.get(rowIndex).getData();
        case 3: return (Object)apuestas.get(rowIndex).getBalioa();
        }
        return null;
    }
}
```

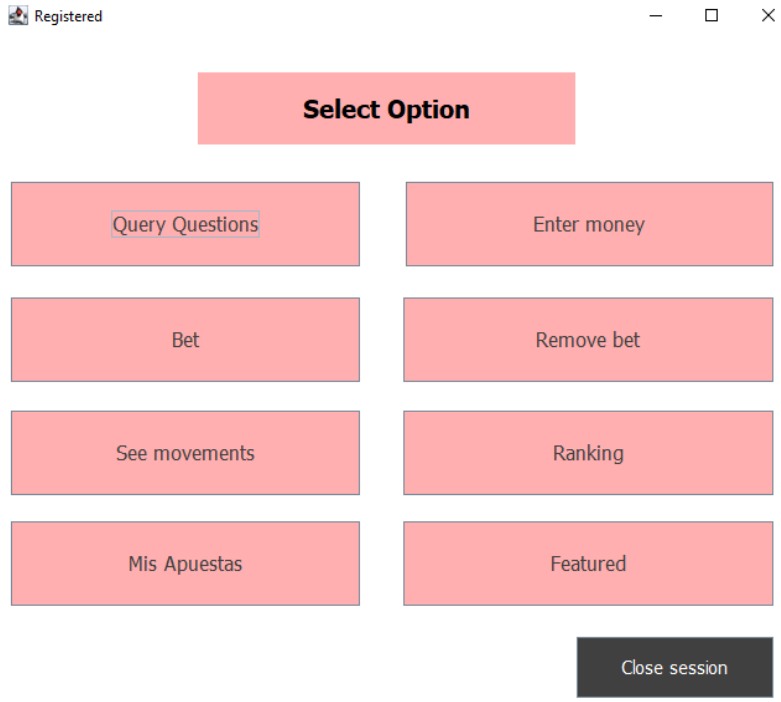
Luego, para crear la GUI que queremos mostrar al usuario para ver sus apuestas, el código es el siguiente:

```
package gui;

import java.awt.BorderLayout;

public class WindowTable extends JFrame {
    private Registered  user;
    private JTable  tabla;

    public WindowTable(Registered user){
        super("Apuestas realizadas por "+   user.getUsername() +":");
        this.setBounds(100, 100,    700,    200);
        this.user = user;

        UserAdapter adapt   =   new UserAdapter(user);

        tabla = new JTable(adapt);
        tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
        //Creamos un JscrollPane    y   le agregamos la JTable
        JScrollPane scrollPane =    new JScrollPane(tabla);
        //Agregamos el  JScrollPane al contenedor
        getContentPane().add(scrollPane,    BorderLayout.CENTER);
    }
}
```

En la implementación de la interfaz principal del usuario, se ha añadido el siguiente trozo de código:

```
public JButton getJButtonVerMisApuestas() {
    if(jButtonVerMisApuestas == null) {
        jButtonVerMisApuestas = new JButton();
        jButtonVerMisApuestas.setFont(new Font("Tahoma", Font.PLAIN, 16));
        jButtonVerMisApuestas.setForeground(Color.DARK_GRAY);
        jButtonVerMisApuestas.setBackground(Color.PINK);
        jButtonVerMisApuestas.setOpaque(true);
        jButtonVerMisApuestas.setText(ResourceBundle.getBundle("Etiquetas").getString("RegisteredGUI.jButtonVerMirApuestas.text"));
        jButtonVerMisApuestas.setBounds(10, 391, 282, 68);
        jButtonVerMisApuestas.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                JFrame a = new WindowTable(user);
                a.setVisible(true);
            }
        });
    }
    return jButtonVerMisApuestas;
}
```

Al hacer login con una cuenta (por ejemplo, ‘andrea’), la interfaz principal para usuarios será la siguiente:



Debido a que la implementación de la clase ‘Registered’ no contempla las apuestas individuales realizadas por el usuario, la interfaz de ‘Mis Apuestas’ contiene únicamente el modelo de tabla:

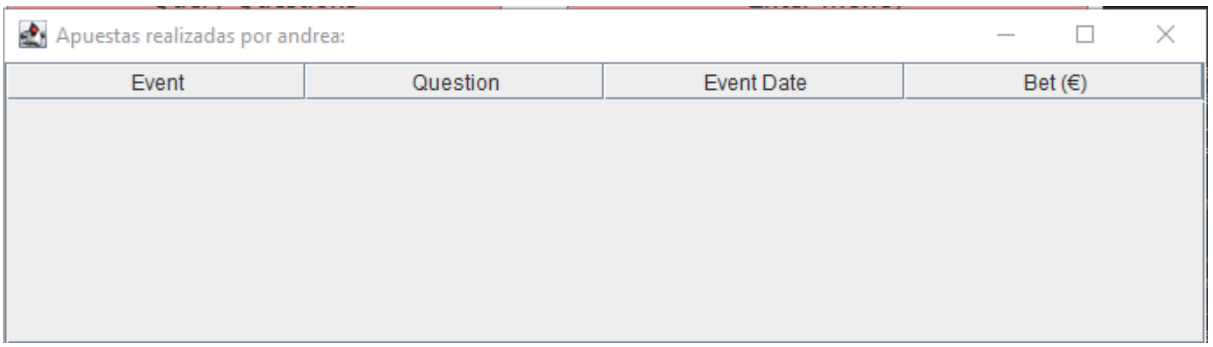
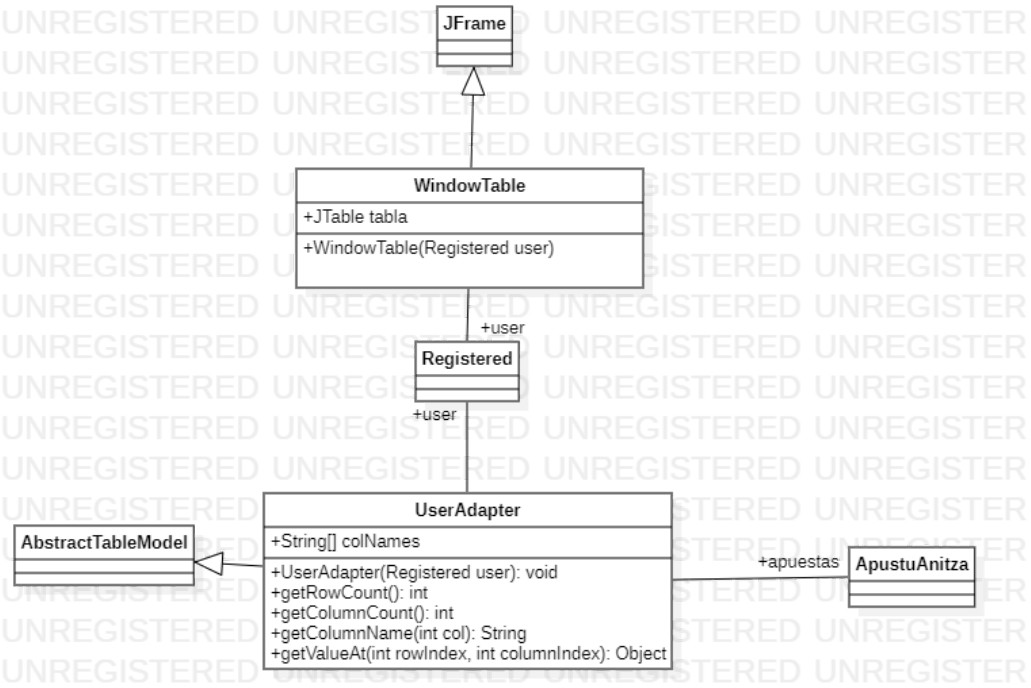


Diagrama UML

El diagrama UML de lo obtenido quedaría tal que así:



Conclusiones

El patrón Adapter facilita el diseño de las interfaces que necesitan mostrar información que está ligada entre varios objetos, de una manera sencilla y legible. El patrón se debe aplicar a clases bien definidas, sin necesidad de modificar ningún aspecto de dichas clases.