# Proyecto Patrones

Contenido

[Proyecto Patrones 1](#_Toc150913450)

[EJERCICIO 3 PATRON ADAPTER 1](#_Toc150913451)

[UML con las nuevas clases 1](#_Toc150913452)

[Código modificado 2](#_Toc150913453)

[Capturas mostrando la ejecución 4](#_Toc150913454)

[EJERCICIO 2 PATRON ITERATOR 6](#_Toc150913455)

[UML con las nuevas clases 6](#_Toc150913456)

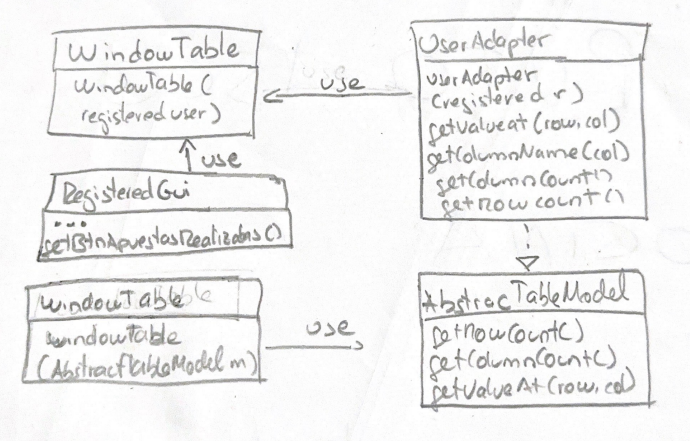
[Código modificado 7](#_Toc150913457)

[Capturas mostrando la ejecución 10](#_Toc150913458)

## EJERCICIO 3 PATRON ADAPTER

Realizado por Iñaki Inda Araguás

### UML con las nuevas clases



### Código modificado

En RegisteredGUI.java he creado un botón para poder acceder a la ventana que muestra la información con todas las apuestas realizadas por un usuario

**private** JButton getBtnApuestasRealizadas() {

**if** (btnAPuestasRealizadas == **null**) {

btnAPuestasRealizadas = **new** JButton(ResourceBundle.*getBundle*("Etiquetas").getString("APREALIZADAS")); //$NON-NLS-1$ //$NON-NLS-2$

btnAPuestasRealizadas.setFont(**new** Font("Tahoma", Font.***PLAIN***, 16));

btnAPuestasRealizadas.setForeground(Color.***DARK\_GRAY***);

btnAPuestasRealizadas.setBackground(Color.***PINK***);

btnAPuestasRealizadas.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

WindowTable vt = **new** WindowTable(user);

vt.setVisible(**true**);

}

});

btnAPuestasRealizadas.setBounds(10, 391, 282, 68);

}

**return** btnAPuestasRealizadas;

Este botón crea una nueva instancia de la clase WindowTable

package gui;

**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));

JScrollPane scrollPane = new JScrollPane(tabla);

getContentPane().add(scrollPane, BorderLayout.CENTER);

}

}

La cual crea una ventana e instancia la clase UserAdapter.

En esta clase en el método getValueAt accedo a las apuestas de un usuario y para cada apuesta que ha realizado recupero los datos

**package** adapter;

**import** java.util.ArrayList;

**import** javax.swing.table.AbstractTableModel;

**import** domain.ApustuAnitza;

**import** domain.Apustua;

**import** domain.Registered;

**public** **class** UserAdapter **extends** AbstractTableModel{

**private** Registered user;

**private** String[] headers = **new** String[]{"Event", "Event Date", "Question", "Bet(€)"};

**private** ArrayList<Apustua> apuestas;

**public** UserAdapter(Registered r) {

**this**.user=r;

apuestas = **new** ArrayList<Apustua>();

**for**(ApustuAnitza a : user.getApustuAnitzak()) {

apuestas.addAll(a.getApustuak());

}

}

@Override

**public** Object getValueAt(**int** row, **int** col) {

Object value = "";

**switch** (col) {

**case** 0:

value = apuestas.get(row).getKuota().getQuestion().getEvent().getDescription();

**break**;

**case** 1:

value = apuestas.get(row).getApustuAnitza().getData();

**break**;

**case** 2:

value = apuestas.get(row).getKuota().getQuestion().getQuestion();

**break**;

**case** 3:

value = apuestas.get(row).getApustuAnitza().getBalioa();

**break**;

}

**return** value;

}

@Override

**public** String getColumnName(**int** col) {

**return** headers[col];

}

@Override

**public** **int** getColumnCount() {

**return** 4;

}

@Override

**public** **int** getRowCount() {

**return** apuestas.size();

}

}

### Capturas mostrando la ejecución

Interfaz de usuario gráfica

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente

## EJERCICIO 2 PATRON ITERATOR

Realizado por Josu Sáez Moreno

### UML con las nuevas clases

Diagrama

Descripción generada automáticamente

### Código modificado

Nuevo método en la lógica de negocio:   
  
Texto

Descripción generada automáticamente

Interfaz Aggregate y clase EventVectorAggregate:  
  
Texto

Descripción generada automáticamente

Interfaz ExtendedIterator:

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Clase EventExtendedIterator:

Texto

Descripción generada automáticamente

Clase IteratorMain:

Texto

Descripción generada automáticamente

### Capturas mostrando la ejecución

Texto

Descripción generada automáticamente con confianza mediaTexto

Descripción generada automáticamente

Texto

Descripción generada automáticamente.

## EJERCICIO 1 FACTORY METHOD

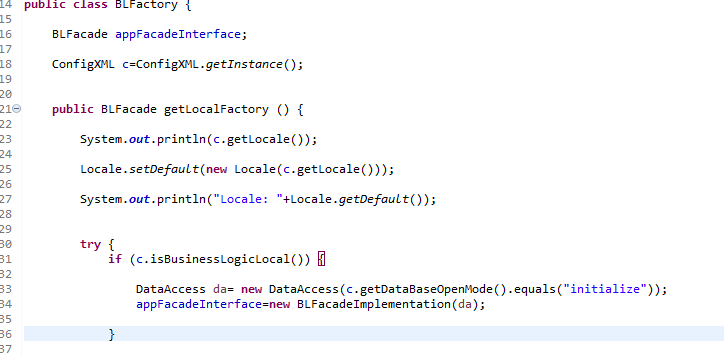
Realizado por Jose Maria Gimeno

Para implementar el este patrón, he creado una nueva clase llamada **BLFactory**.

Esta nueva clase lo que hace es ver si se quiere crear una BLFacade Local o una BLFacade Distribuida por Servicios Web.

El creator es BLFactory, el product es BLFacade y el ConcreteProduct es la implementación concreta de la factoría que decide qué implementación de BLFacade crear.

He creado el método **getLocalFactory(),** éste pregunta si el objeto creado c(que es una instancia de ConfigXML ) es local o no.



Desde la clase ApplicationLauncher() llamamos a la clase BLFactroy para que ella decida que Desde la clase ApplicationLauncher() llamamos a la clase BLFactroy para que ella decida que instancia de la BLFacace crear(local o distribuida)

