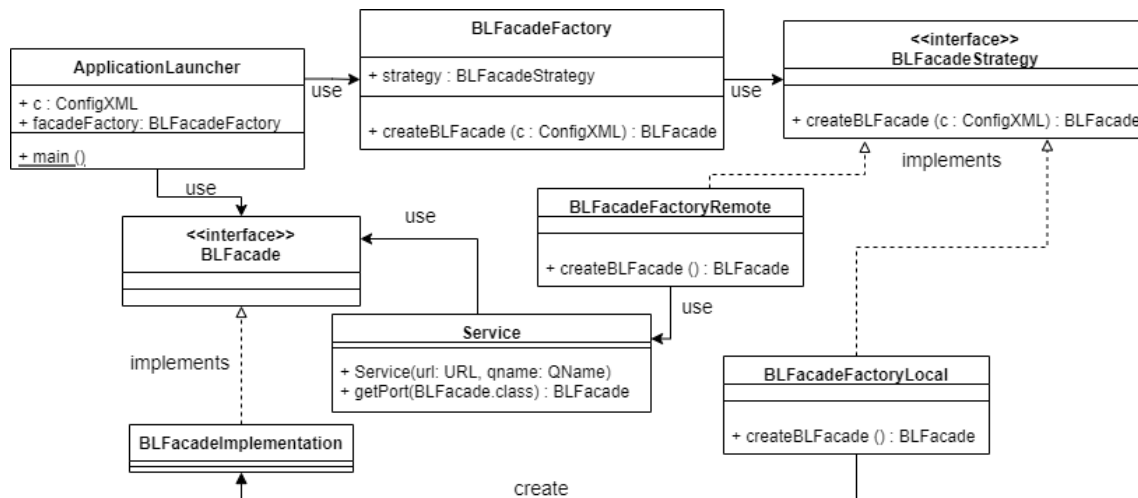


Autores: Nestor Guimerans, Jaione Gonzalez, Abderraouf Khedidji (grupo 1.4)

Link al repositorio: <https://github.com/JaioneGonzalez/betsProiektua>

1. PATRÓN FACTORY METHOD

UML:



CÓDIGO MODIFICADO:

```
public class BLFacadeFactoryLocal implements BLFacadeStrategy{

    @Override
    public BLFacade createBLFacade(ConfigXML c) {
        // TODO Auto-generated method stub
        DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
        return new BLFacadeImplementation(da);
    }

}
```

```
public class BLFacadeFactoryRemote implements BLFacadeStrategy{

    @Override
    public BLFacade createBLFacade(ConfigXML c) {
        String serviceName= "http://"+c.getBusinessLogicNode()+":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName()+"?wsdl";

        //URL url = new URL("http://localhost:9999/ws/ruralHouses?wsdl");
        URL url = null;
        try {
            url = new URL(serviceName);
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //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);
        return service.getPort(BLFacade.class);
    }

}
```

```
public interface BLFacadeStrategy {
    public BLFacade createBLFacade(ConfigXML c);
}
```

```
public class BLFacadeFactory {
    BLFacadeStrategy strategy;

    public BLFacadeFactory(ConfigXML c) {
        if (c.isBusinessLogicLocal()) {
            strategy = new BLFacadeFactoryLocal();
        } else {
            strategy = new BLFacadeFactoryRemote();
        }
    }

    public BLFacade createBLFacade(ConfigXML c) {
        return strategy.createBLFacade(c);
    }
}
```

Cambios realizados en la clase *ApplicationLauncher* (dentro del try):

```
BLFacade appFacadeInterface;
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
BLFacadeFactory factory = new BLFacadeFactory(c);
appFacadeInterface = factory.createBLFacade(c);
MainGUI.setBusinessLogic(appFacadeInterface);
```

COMENTARIOS:

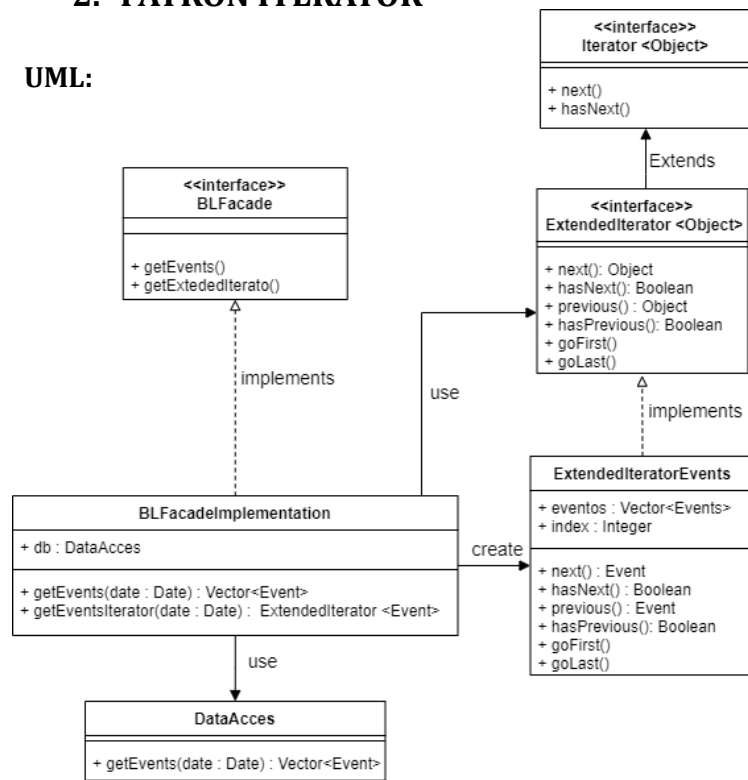
Le hemos quitado la responsabilidad de decidir cómo acceder a la lógica de negocio a *ApplicationLauncher*, y la hemos separado en tres clases distintas.

Por un lado están *BLFacadeFactoryLocal* y *BLFacadeFactoryRemote*. Cada una de estas accede a un objeto de lógica de negocio de una forma distinta (local o remota, según indican sus nombres), y la última clase (*BLFacadeFactory*), se encarga de decidir cuál de las dos utilizar.

De esta forma, la clase *ApplicationLauncher* tiene un objeto *BLFacadeFactory*, independientemente de cómo se acceda a la lógica de negocio.

2. PATRÓN ITERATOR

UML:



CÓDIGO MODIFICADO:

```

public interface ExtendedIterator<Object> extends Iterator<Object> {

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

```

```

public class ExtendedIteratorEvents implements ExtendedIterator {
    private Vector<Event> eventos;
    private int index;
    public ExtendedIteratorEvents(Vector<Event> eventos) {
        this.eventos = eventos;
        index = 0;
    }
    @Override
    public boolean hasNext() {
        try {
            if (eventos.get(index) != null) {
                return true;
            }
            return false;
        } catch (Exception e) {
            return false;
        }
    }

    @Override
    public Event next() {
        index += 1;
        return eventos.get(index-1);
    }

    @Override
    public Event previous() {
        index -= 1;
        return eventos.get(index+1);
    }
}

```

```

@Override
public boolean hasPrevious() {
    return hasNext();
}

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

@Override
public void goLast() {
    index = eventos.size()-1;
}

```

```

public class mainIterator {

    public static void main(String[] args) {
        int isLocal = 1;
        BLFacade blFacade = new BLFacadeImplementation();
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Date date;
        try {
            date = sdf.parse("17/12/2022");
            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");
        }
    }
}

```

En la clase *BLFacadeImplementation*:

```

@WebMethod
public ExtendedIterator<Event> getEventsIterator(Date date){
    return new ExtendedIteratorEvents(getEvents(date));
}

```

CAPTURA DE LA EJECUCIÓN:

```

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

```

COMENTARIOS:

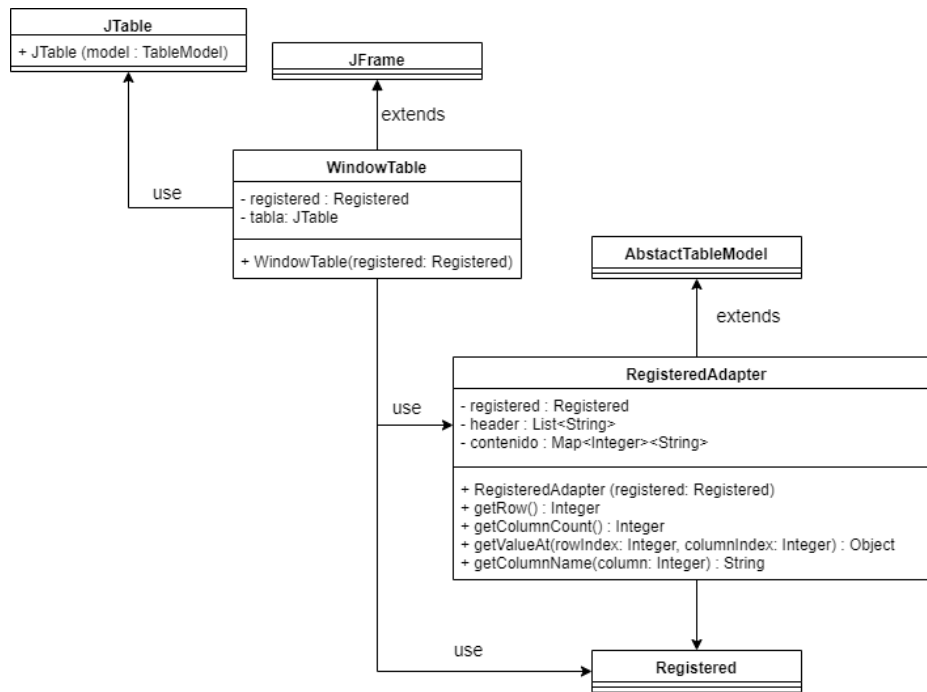
Hemos creado la interfaz *ExtendedIterator* para luego crear la clase *ExtendedIteratorEvents*, la cual implementa esta interfaz.

Esta clase se comporta como un iterador normal con las funciones del *hasNext()* y *next()* con el añadido de que también puede correr hacia atrás (usando las funciones de *previous()* y *hasPrevious()*) y nos podemos posicionar al comienzo y al final de la lista (usando las funciones de *goFirst()* y *goLast()*).

Este iterador usa un vector de eventos, pero usando la interfaz, se puede utilizar con otros tipos de elementos sin necesidad de hacer muchos cambios.

3. PATRÓN ADAPTER

UML:



CÓDIGO MODIFICADO:

```
public class WindowTable extends JFrame {
    private Registered register;
    private JTable tabla;
    public WindowTable(Registered register) {
        super("Apuestas realizadas por "+register.getUsername()+":");
        this.setBounds(100,100,700,200);
        this.register = register;
        RegisteredAdapter adapt = new RegisteredAdapter(register);
        tabla = new JTable (adapt);
        tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
        JScrollPane scrollPane = new JScrollPane(tabla);
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }
}
```

```

public class RegisteredAdapter extends AbstractTableModel{
    private Registered register;
    private String[] header = {"Event", "Question", "Event Date", "Bet(€)"};

    private Map<Integer, List<String>> contenido = new HashMap<Integer, List<String>>();
    int i = 0;
    public RegisteredAdapter(Registered register) {
        this.register = register;

        for (ApustuAnitza apu:register.getApustuAnitzak()) {
            for (Apustua ap: apu.getApustuak()) {

                Quote quota = ap.getKuota();
                Question question = quota.getQuestion();
                Event eventoNuevo = question.getEvent();
                List<String> fila = new ArrayList<String>();
                fila.add(eventoNuevo.getDescription());
                fila.add(question.getQuestion());
                fila.add(eventoNuevo.getEventDate().toString());
                fila.add(quota.getQuote().toString());

                contenido.put(i, fila);
                i++;
            }
        }
    }

    @Override
    public int getRowCount() {
        return i;
    }

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

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        return contenido.get(rowIndex).get(columnIndex);
    }

    @Override
    public String getColumnName(int column) {
        return header[column];
    }
}

```

Cambios en la clase *MainUserGUI* (dentro del método *getPanel()*):

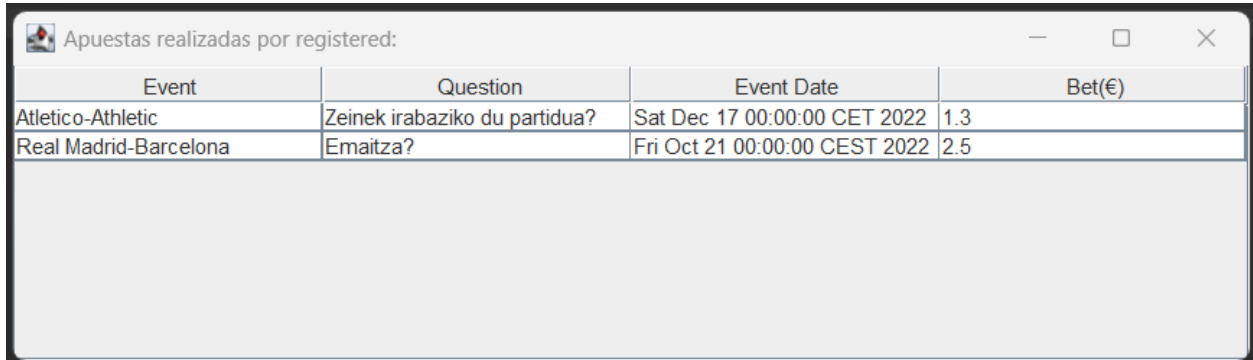
```

JButton btnAdapter = new JButton(ResourceBundle.getBundle("Etiquetas").getString("MainUserGUI.btnNewButton.text"));
btnAdapter.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e){
        try {
            BLFacade blFacade= new BLFacadeImplementation();
            Registered register = new Registered("registered","123",1234);
            Registered register2 = blFacade.findUser(register);

            WindowTable vt = new WindowTable(register2);
            vt.setVisible(true);
        }
        catch (Exception e1) {
            e1.printStackTrace();
        }
    }
});
panel.add(btnAdapter);

```


CAPTURA DE LA EJECUCIÓN:



Event	Question	Event Date	Bet(€)
Atletico-Athletic	Zeinek irabaziko du partidua?	Sat Dec 17 00:00:00 CET 2022	1.3
Real Madrid-Barcelona	Eraitza?	Fri Oct 21 00:00:00 CEST 2022	2.5

COMENTARIOS:

Hemos creado la clase *RegisteredAdapter* para aprovecharnos de todas las ventajas que nos proporciona la clase *AbstractTableModel*, de la cual extiende. Y con nuestro atributo *Registered* obtenemos todos los datos que deseamos reflejar en nuestra tabla *JTable*, devolviéndolo con el método sobrescrito *getValueAt()* sin necesidad de modificar nada en la dicha clase *Registered*.

Además hemos implementado la clase *WindowTable* que es la encargada de inicializar el objeto *Registered* para introducirlo en *RegisteredAdapter* y este último en nuestra tabla de *JTable*.