[PATRONES DE DISEÑO]

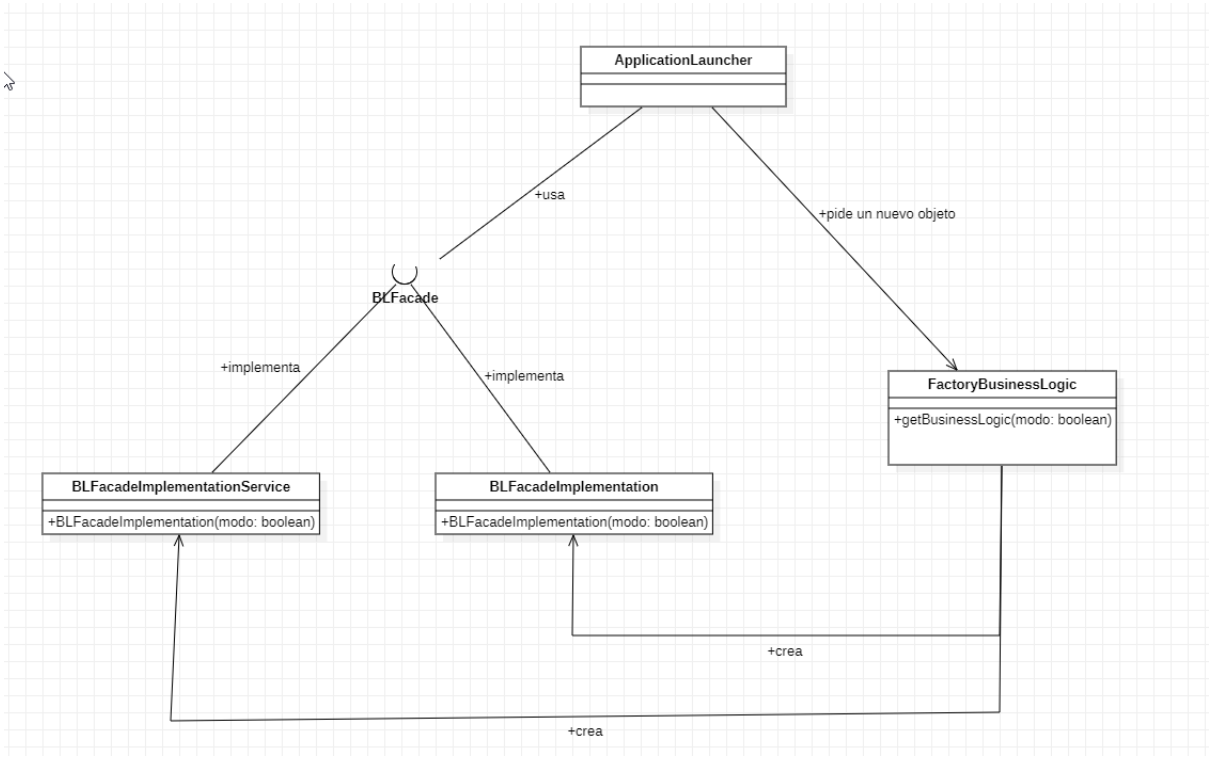
Ingeniería del Software II
Grado en Ingeniería Informática
Facultad de Informática
Universidad del País Vasco

Martinez Amunarriz, Pablo
Silva Alonso, Raúl

14/11/2022

Patrón Factory Method

a) Diagrama UML extendido, indicando los cambios realizados y la finalidad de cada clase/interfaz.



-Application Launcher: es la clase “Cliente” que solicita la creación de un nuevo objeto. He quitado el if else que se usaba para diferenciar qué lógica que se quería crear, además de borrar la creación de estas. De este modo, ApplicationLauncher crea y llama a la nueva clase FactoryBusinessLogic con el valor que quiera (false si quiere crear una lógica local y true si quiere una lógica distribuida a través de servicios web).

-FactoryBusinessLogic: es la clase “Creator” que se ocupa de la funcionalidad que tenía antes el ApplicationLauncher, es decir, tiene un método llamado getBusinessLogic el cual dependiendo del parámetro que le pasen creará un tipo de lógica o otra y la devolverá(BLFacadeImplementation o BLFacadeImplementationService).

-BLFacadeImplementation: es una de las clases “ConcreteProduct”. Esta clase implementa la interfaz BLFacade. Además, en esta clase he puesto una sola constructora la cual tiene un parámetro booleano que se le pasa a la clase DataAccess para inicializarla.

-BLFacadeImplementationService: creo que esta es la otra las clases “ConcreteProduct” (la cual no está creada, pero diría que es la clase que representa la lógica distribuida a través de servicios web).

-BLFacade: es la interfaz “Product”. Esta interfaz la van a implementar todas los tipos de lógica de negocio que haya. Está creada para cumplir con el principio de abierto a la extensión y cerrado a la modificación.

b) El código modificado:

Clase BLFacadeImplementation

```java
package businessLogic;
import java.util.Collection;

/**
 * It implements the business logic as a web service.
 */
@WebService(endpointInterface = "businessLogic.BLFacade")
public class BLFacadeImplementation  implements BLFacade {

    DataAccess dbManager;



    public BLFacadeImplementation(boolean modo) {
        dbManager = new DataAccess(modo);
        dbManager.initializeDB();
        dbManager.close();
    }
```

He creado la nueva constructora.

Clase BusinessLogicServer

```java
try {

    try{

        if (!c.isDatabaseLocal()) {
            System.out.println("\nWARNING: Please be sure ObjectdbManagerServer is launched\n          in machine: "+c.getDatab
        }

        service= "http://"+c.getBusinessLogicNode() +":"+ c.getBusinessLogicPort()+"/ws/"+c.getBusinessLogicName();

        Endpoint.publish(service, new BLFacadeImplementation(true));
```

Al crear una nueva clase BLFacadeImplementation le paso un parámetro necesario y true para crear una lógica distribuida a través de servicios web.

Clase FactoryBusinessLogic

```java
package businessLogic;

import java.net.MalformedURLException;

public class FactoryBusinessLogic {

    public BLFacade getBusinessLogic(boolean modo) {
        BLFacade interfaz = null;

        if (modo) {
            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);
                interfaz = service.getPort(BLFacade.class);

            }catch (MalformedURLException e) {
                System.out.println("Error al crear la logica de negocio remota");
            }

        }else {
            interfaz = new BLFacadeImplementation(false);
        }
        return interfaz;
    }


}
```

La nueva clase “Creator”.
Clase ApplicationLauncher

```
    public static void main(String[] args) {

        ConfigXML c=ConfigXML.getInstance();

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

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

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

        MainGUI a=new MainGUI();
        a.setVisible(false);

        MainUserGUI b = new MainUserGUI();
        b.setVisible(true);


        try {

            boolean modo = false;

            FactoryBusinessLogic factory = new FactoryBusinessLogic();
            BLFacade interfaz = factory.getBusinessLogic(modo);


//          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");

            MainGUI.setBussinessLogic(interfaz);


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

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

Como se puede ver se crea una instancia de FactoryBusinessLogic y luego la usa para llamar a su método getBusinessLogic con el parámetro que desea y recibir la nueva interfaz.

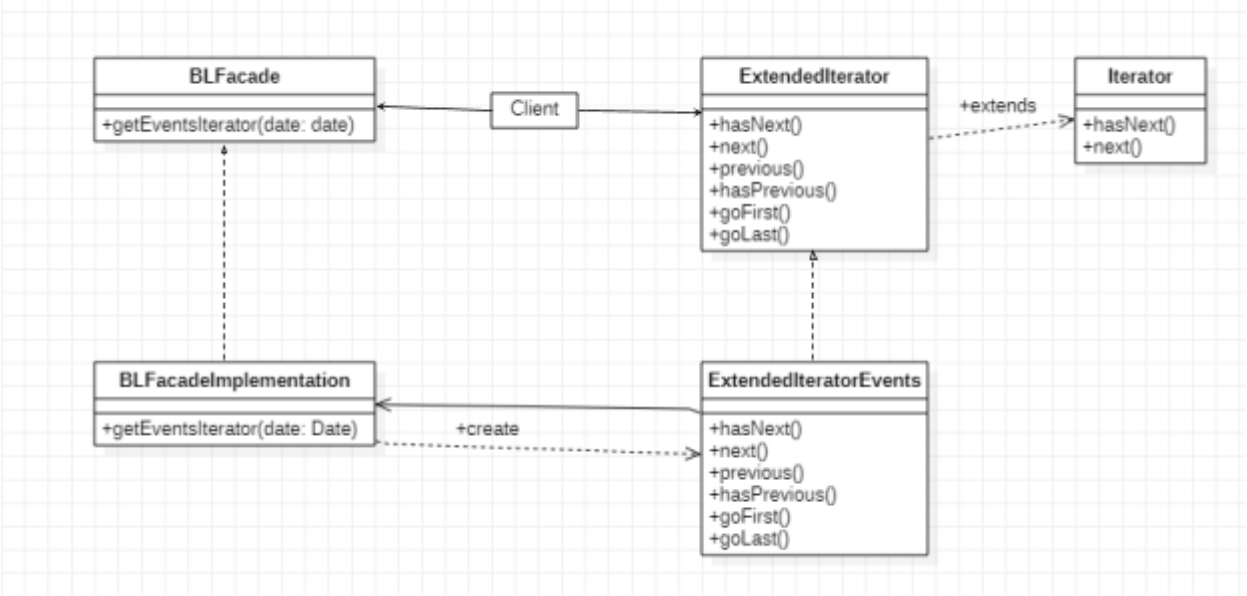
Clase main

```
import java.text.ParseException;

public class main {

    public static void main(String[] args) {
        BLFacade blFacade = new FactoryBusinessLogic().getBusinessLogic(false);
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Date date;
        try {
```

Esta foto es un ejemplo de que en las clases en las que creaba una instancia la de cualquier lógica ha sido cambiada y ahora hay que realizar esta llamada pasando el parámetro que desees dependiendo de la lógica.

Patrón Iterator

a) Diagrama UML extendido, indicando los cambios realizados y la finalidad de cada clase/interfaz.



He creado un paquete llamado “iterator” en el cual he implementado las siguientes clases/interfaces.

- La interfaz ExtendedIterator que es una hija de la clase Iterator y que implementa a parte de los metodos de la clase iterator (hasNext() y next()) los métodos: previous(), que devuelve el elemento actual y se mueve una posición hacia atrás; hasPrevious(), que devuelve true si tiene un elemento en la posición anterior a la actual; goFirst(), que se coloca en el primer elemento; y goLast(), que se coloca en el último elemento.
- La clase ExtendedIteratorEvents que es la clase que implementa todos los métodos de la interfaz ExtendedIterator. Esta clase tiene un índice para saber la posición en el vector y un vector que contendrá eventos. A parte de implementar todos los métodos de la interfaz también hay un constructor al cual se le pasa el vector de eventos y se inicializa el índice a 0.
- En la interfaz BLFacade se declara la función getEventsIterator(date:Date).
- En la clase BLFacadeImplementation se implementa la función getEventsIterator(date:Date) donde creará un objeto de ExtendedIteratorEvents, pasándole como parámetro el método getEvents(date), que este devuelve un vector de eventos.

b) El código modificado:

<u>Interfaz ExtendedIterator</u>

```java
package iterator;

import java.util.Iterator;

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

    //Devuelve el elemento actual y va haca atras
    public Object previous();

    //true  si tiene un elemento anterior
    public boolean hasPrevious();

    //Se coloca en el primer elemento
    public void goFirst();

    //Se coloca en el último elemento
    public void goLast();
}
```

Clase ExtendedIteratorEvents

```java
package iterator;

import java.util.Vector;

public class ExtendedIteratorEvents implements ExtendedIterator {

    private Vector<Event> events;
    private int i;

    public ExtendedIteratorEvents(Vector<Event> events) {
        this.events = events;
        i = 0;
    }

    @Override
    public boolean hasNext() {
        try {
            if(events.get(i) != null)
                return true;
            return false;
        }catch(Exception e) {
            return false;
        }
    }

    @Override
    public Object next() {
        i++;
        return events.get(i-1);
    }

    @Override
    public Object previous() {
        i--;
        return events.get(i+1);
    }

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

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

    @Override
    public void goLast() {
        i = events.size()-1;
    }

}
```

Interfaz BLFacade

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

Esta línea la hemos añadido al final.

Clase BLFacadeImplementation

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

Esta línea la hemos añadido al final.

c) Captura de imagen mostrando la ejecución:

Eventos del día 17/12/2022

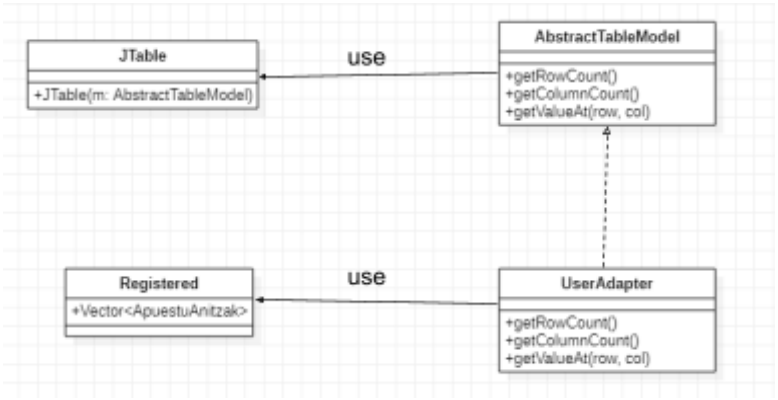
```
>> DataAccess: getEvents
1;Atletico-Athletic
2;Eibar-Barcelona
3;Getafe-Celta
4;Alaves-Deportivo
5;Espanol-Villareal
6;Las Palmas-Sevilla
7;Malaga-Valencia
8;Girona-Leganes
9;Real Sociedad-Levante
10;Betis-Real Madrid
22;LA Lakers-Phoenix Suns
23;Atlanta Hawks-Houston Rockets
24;Miami Heat-Chicago 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
```

Eventos del día 01/12/2022

```
>> DataAccess: getEvents
11;Atletico-Athletic
12;Eibar-Barcelona
13;Getafe-Celta
14;Alaves-Deportivo
15;Espanol-Villareal
16;Las Palmas-Sevilla
25;Boston Celtics-Memphis Grizzlies
26;Nadal-Alcaraz
DataBase closed

_____________________
RECORRIDO       HACIA   ATRÁS
26;Nadal-Alcaraz
25;Boston Celtics-Memphis Grizzlies
16;Las Palmas-Sevilla
15;Espanol-Villareal
14;Alaves-Deportivo
13;Getafe-Celta
12;Eibar-Barcelona
11;Atletico-Athletic


_____________________
RECORRIDO       HACIA   ADELANTE
11;Atletico-Athletic
12;Eibar-Barcelona
13;Getafe-Celta
14;Alaves-Deportivo
15;Espanol-Villareal
16;Las Palmas-Sevilla
25;Boston Celtics-Memphis Grizzlies
26;Nadal-Alcaraz
```

Patrón Adapter

a) Diagrama UML extendido, indicando los cambios realizados y la finalidad de cada clase/interfaz.



-**JTable:** Es una clase que hemos creado que es hijo de la clase AbstractTableModel que representa una tabla que nos informa sobre todas las apuestas realizadas por el usuario.
-**AbstractTableModel:** Es una clase que usa el JTable para su método.
-**Registered:** Es la clase que guarda los usuarios registrados.
-**UserAdapter:** Es una clase que implementa la clase de AbstractTableModel e implementa sus métodos. De este modo, registered puede hacer uso de estos métodos.

b) El código modificado:

```
package gui;

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import domain.Registered;
import domain.UserAdapter;

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);
    }
}
```

https://github.com/paabloomaartinez/Bets22_PabloRaul

```java
package domain;

import java.util.List;

public class UserAdapter extends AbstractTableModel {
    private final Vector<ApustuAnitza> ListaApuestas;
    private Registered usuario;
    private String[] nombreDeLasColumnas = new String[] {"Event", "Question", "Event Date", "Bet (€)"};

    public UserAdapter(Registered user) {
        ListaApuestas = new Vector<ApustuAnitza>(user.getApustuAnitzak());
        this.usuario = user;
    }

    public int getRowCount() {
        return ListaApuestas.size();
    }

    public int getColumnCount() {
        return 4;
    }

    public String getColumnName(int col) {
        return nombreDeLasColumnas[col];
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        switch(columnIndex) {
        case 0: return (Object)ListaApuestas.get(rowIndex).getApustuak();
        case 1: return (Object)ListaApuestas.get(rowIndex);
        case 2: return (Object)ListaApuestas.get(rowIndex).getData();
        case 3: return (Object)ListaApuestas.get(rowIndex).getBalioa();
        }
        return null;
    }
}
```

c) Captura de imagen mostrando la ejecución: