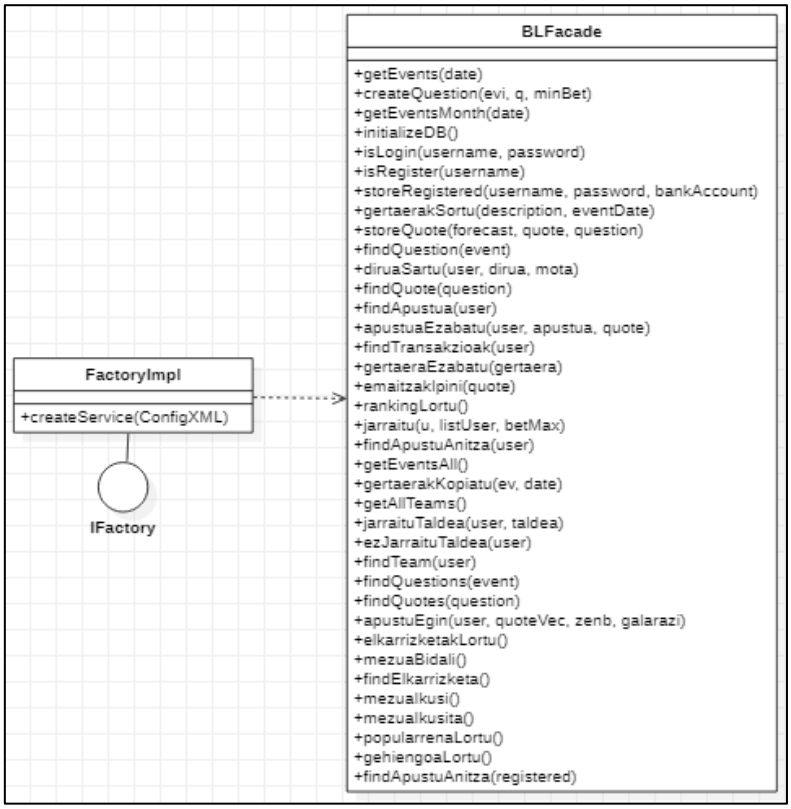
# ProyectoISCI patrones de diseño

## Tabla de contenido

# 1. Patrón Factory Method

## 1.1. Diagrama UML extendido




## 1.2. Finalidad de cada clases e interfaces implementadas

### *1.2.1. FactoryImpl*

Creamos la clase FactoryIml que implementa la interfaz IFactory. En esta clase creamos el método createService() recibiendo como parámetro de entrada un objeto ConfigXML (el que se crea en el main() de la clase AplicationLauncher). Hemos decidido pasarlo por parámetro puesto que vamos a precisar de sus métodos tal y como se utilizaban en el AplicationLauncher. De esta manera, mediante el objeto ConfigXML (c), podemos controlar si la instancia de BLFacadeImplementation es local o remota.

En el caso de local, devolvemos una nueva instancia de BLFacadeImplementation (con la instancia de la DataAccess como parámetro).

Para este segundo caso, hemos añadido un try-catch de manera que trate de instanciar tanto la URL como el QName y cree el servicio asociado a dichos valores. En el hipotético caso de que no se creará correctamente el servicio, el método createService() devolverá null.

### *1.2.2. IFactory*

En esta interfaz simplemente creamos el método createService(createService) que definimos en la clase FactoryIml.

## 1.3. Código modificado

### *1.3.1. FactoryImpl*

```java
package factory;

import java.net.MalformedURLException;

public class FactoryImpl implements IFactory{

    @Override
    public BLFacade createService(ConfigXML c) {
        if (c.isBusinessLogicLocal()) {
            DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));

            return new BLFacadeImplementation(da);
        }else {
            String serviceName = "http://" + c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort() + "/ws/"
                    + c.getBusinessLogicName() + "?wsdl";
            URL url;
            try {
                url = new URL(serviceName);
                QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
                Service service = Service.create(url, qname);
                return service.getPort(BLFacade.class);
            } catch (MalformedURLException e) {
                e.printStackTrace();
                return null;
            }
        }
    }
}
```

### *1.3.2. IFactory*

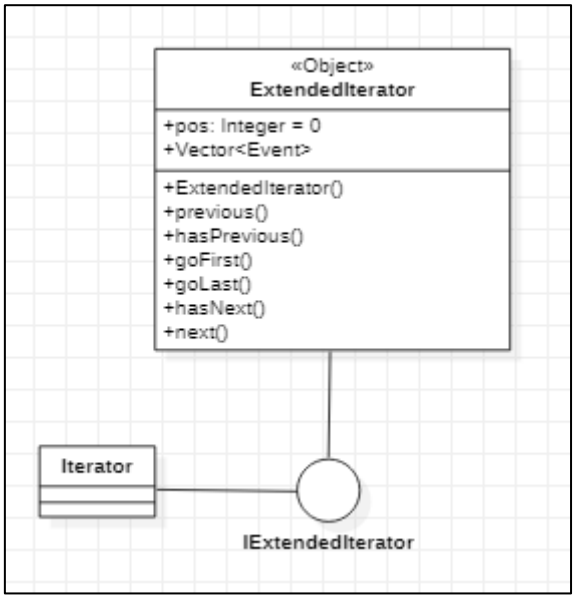
```java
package factory;

import businessLogic.BLFacade;

public interface IFactory {
    public BLFacade createService(ConfigXML c);
}
```

### *1.3.3. AplicationLauncher*

```java
        try {

            IFactory fac = new FactoryImpl();
            BLFacade appFacadeInterface = fac.createService(c);
//          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.isBusinessLogicLocal()) {
//
//              //In this option the DataAccess is created by FacadeImplementationWS
//              //appFacadeInterface=new BLFacadeImplementation();
//
//              //In this option, you can parameterize the DataAccess (e.g. a Mock DataAccess object)
//
//              DataAccess da= new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
//              appFacadeInterface=new BLFacadeImplementation(da);
//
//
//          }
//
//          else { //If remote
//
//               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);
//          }
```

# 2. Patrón Iterator

## 2.1. Diagrama UML extendido




## 2.2. Finalidad de cada clases e interfaces implementadas

### *2.2.1. ExtendedIterator*

La clase ExtendedIterator implementa la interfaz IExtendedIterator. En primer lugar, creamos una variable que contenga la posición del iterador, además de un vector de eventos con el que trabajaremos a continuación.

Esta clase se encarga de sobrescribir los métodos next(), hasNext(), previous(), hasPrevious(), goFirst() y goLast(). Next() y previous() se encargan de mover una posición hacia delante y hacia detrás respectivamente devolviendo a continuación el objeto de la posición actual.

A su vez, hasNext() y hasPrevious() nos indican si el objeto de la posición siguiente y anterior respectivamente son distintos de null, devolviendo por tanto un booleano. En el caso de hasPrevious() precisamos de un try-catch para controlar el caso de encontrarnos en la primera posición. En hasNext() no es necesario porque comprobamos que la posición actual sea menor que la longitud.

Por último, goFirst() y goLast() sitúan la posición al comienzo y al final del iterador respectivamente.

### *2.2.2. IExtendedIterator*

En esta interfaz en primer lugar extendemos la clase Iterator<Object> para poder trabajar con iteradores. A continuación, definimos los métodos previous(), hasPrevious(), goFirst() y goLast(). En este caso no precisamos de definir next() y hasNext() puesto que ya los contiene la clase Iterator.

## 2.3. Código modificado

### *2.3.1. ExtendedIterator*

```java
package iterator;

import java.util.Iterator;
import java.util.Vector;

import domain.Event;

public class ExtendedIterator<Object> implements IExtendedIterator{
    int pos = 0;
    Vector<Event> events;

    public ExtendedIterator(Vector<Event> vector) {
        this.events = vector;
    }
    @Override
    public Object previous() {
        Event ev = events.get(pos);
        pos--;
        return (Object) ev;
    }

    @Override
    public boolean hasPrevious() {
        try {
            int aux = pos-1;
            Event ev = events.get(aux);
            return ev!=null;
        }catch(Exception e) {
            return false;
        }

    }
```

```java

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

    @Override
    public void goLast() {
        pos = events.size();
    }

    @Override
    public boolean hasNext() {
        return pos<events.size();
    }

    @Override
    public Object next() {
        // TODO Auto-generated method stub
        Event ev = events.get(pos);
        pos++;
        return (Object) ev;
    }

}
```

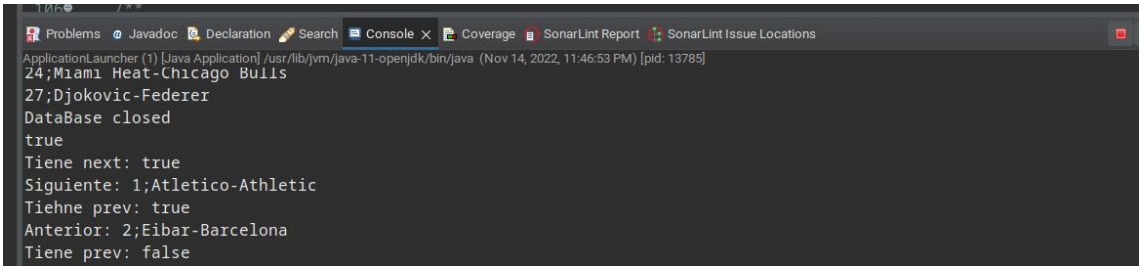
### *2.3.2. IExtendedIterator*

```java
package iterator;

import java.util.Iterator;

public interface IExtendedIterator extends Iterator<Object>{
    public Object previous();
    public boolean hasPrevious();
    public void goFirst();
    public void goLast();
}
```

#### 2.3.3. AplicationLauncher

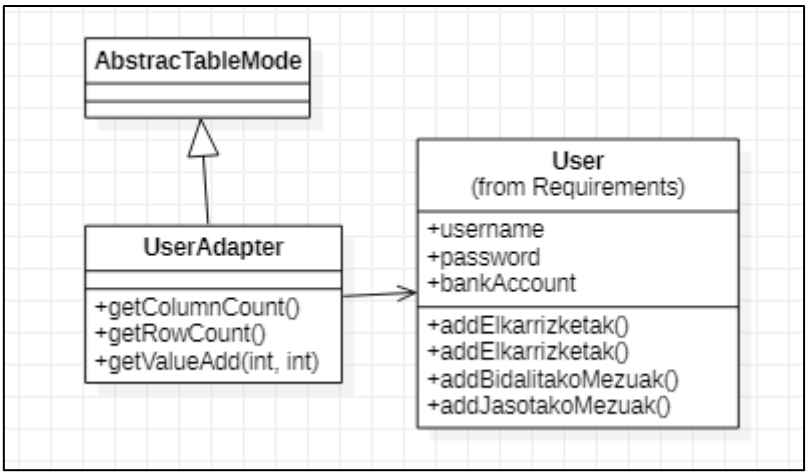
```
86          MainGUI.setBussinessLogic(appFacadeInterface);
87
88          Date date = null;
89          SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
90          try {
91              date = sdf.parse("17/12/2022");
92          } catch (ParseException e) {
93              e.printStackTrace();
94          }
95          IExtendedIterator iter = appFacadeInterface.getEventsIterator(date);
96          System.out.println("Tiene next: " + iter.hasNext());
97          System.out.println("Siguiente: " + iter.next());
98          System.out.println("Tiehne prev: " + iter.hasPrevious());
99          System.out.println("Anterior: " + iter.previous());
100         System.out.println("Tiene prev: " + iter.hasPrevious());
101
```

### 2.4. Ejecución




## 3. Patrón Adapter

### 3.1. Diagrama UML extendido




### 3.2. Finalidad de cada clases e interfaces implementadas

Adaptar el tipo de User a una tabla con las apuestas que tenga dicho usuario.

### 3.3. Código modificado

No hemos conseguido modificar el código de manera que nos funcione.

### 3.4. Ejecución

No nos funciona la ejecución.