Universidad del País Vasco

Facultad de Informática

Ingenieria de Software II

Proyecto Patrones

Francisco Cagnoli - 1128397

Andrés Juan - 1128392

Hernán Yáñez - 1128319

Github

2022

1. Factory

Se creó la clase BLFactory que se encarga de crear y devolver al ApplicationLauncher la implementación de BLFacade, dependiendo de la que quiera utilizar (actualmente decide entre remoto y local).

Código

Version Anterior

src/main/java/gui/ApplicationLauncher.java

```
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 {
        BLFacade appFacadeInterface;
        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        if (c.isBusinessLogicLocal()) {
               DataAccess da= newDataAccess(c.getDataBaseOpenMode().equals("initialize"));
               appFacadeInterface=new BLFacadeImplementation(da);
        }else {
            String serviceName= "http://"+c.getBusinessLogicNode()
                                +":"+c.getBusinessLogicPort()+"/ws/"
                                +c.getBusinessLogicName()+"?wsdl";
            URL url = new URL(serviceName);
            QName qname = new QName("http://businessLogic/","BLFacadeImplementationService");
            Service service = Service.create(url, qname);
            appFacadeInterface = service.getPort(BLFacade.class);
        }
        MainGUI.setBussinessLogic(appFacadeInterface);
    }catch (Exception e) {
        a.jLabelSelectOption.setText("Error: "+e.toString());
        a.jLabelSelectOption.setForeground(Color.RED);
        System.out.println("Error in ApplicationLauncher: "+e.toString());
    }
}
```

Version Nueva

src/main/java/gui/ApplicationLauncher.java

```
public static void main(String[] args) {
       ConfigXML c = ConfigXML.getInstance();
       BLFactory blFactory = new BLFactory(c);
       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 {
          BLFacade appFacadeInterface;
          UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
          if (c.isBusinessLogicLocal()) {
              appFacadeInterface = blFactory.createBLFacade(BL_IMPLEMENTATIONS.LOCAL);
          } else { // If remote
              appFacadeInterface = blFactory.createBLFacade(BL_IMPLEMENTATIONS.REMOTE);
          }
          MainGUI.setBussinessLogic(appFacadeInterface);
       } catch (Exception e) {
       a.jLabelSelectOption.setText("Error: " + e.toString());
       a.jLabelSelectOption.setForeground(Color.RED);
       System.out.println("Error in ApplicationLauncher: " + e.toString());
   }
}
```

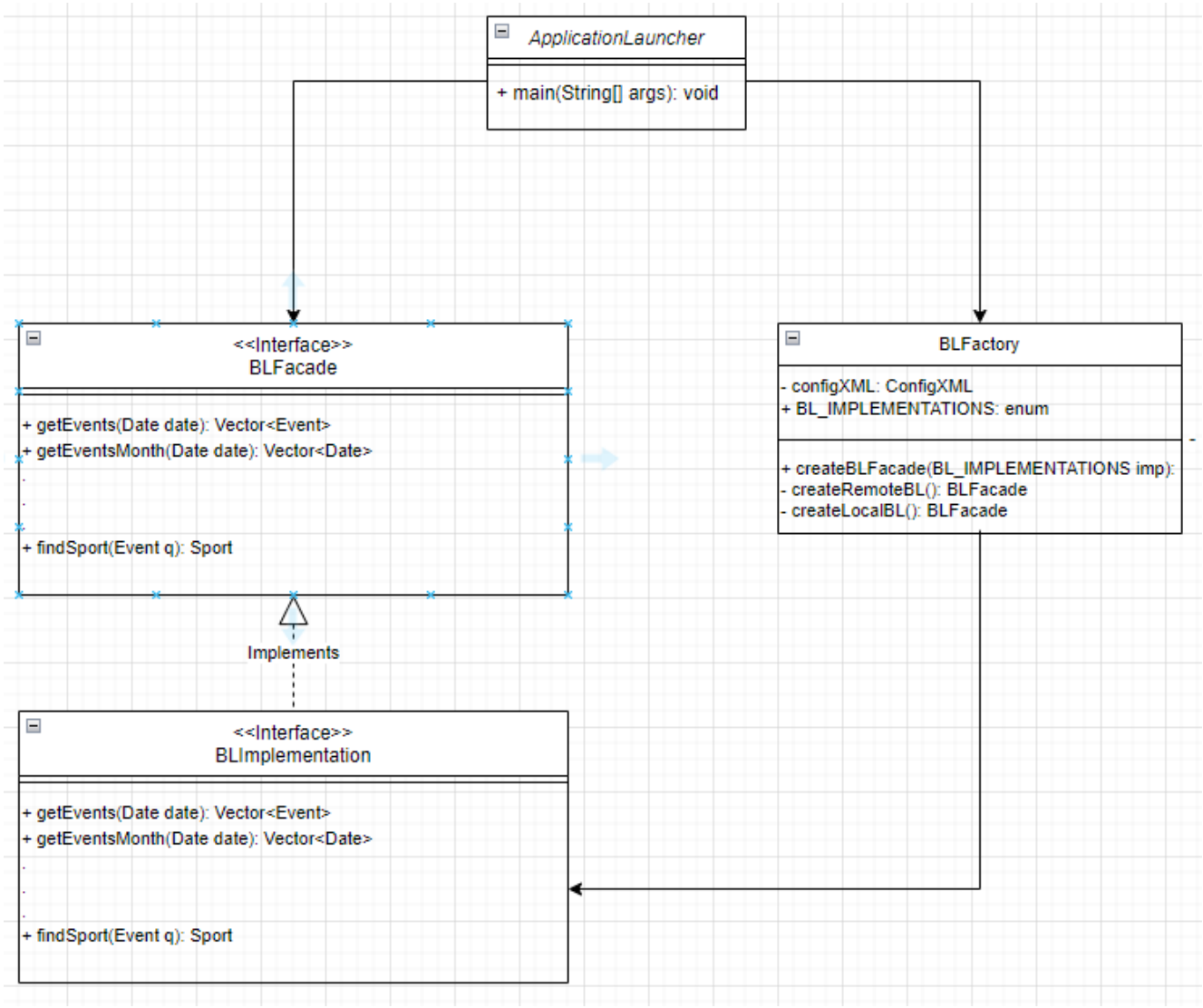
Aquí la implementación se deriva a la Factory en vez de crearlo en el Main. El enum BL_IMPLEMENTATIONS nos ayuda a no equivocarnos al elegir la implementacion.

src/main/java/businessLogic/BLFactory.java

```
public class BLFactory {
	public enum BL_IMPLEMENTATIONS {
		LOCAL, REMOTE
	}
	private ConfigXML c;
	public BLFactory (ConfigXML config) {
		this.c = config;
	}
	public BLFacade createBLFacade(BL_IMPLEMENTATIONS imp) {
		if (imp == null)
			return null;

		switch (imp) {
		case LOCAL:
			return createLocalBL();
		case REMOTE:
			return createRemoteBL();
		default:
			throw new IllegalArgumentException("Unknown option " + imp);
		}
	}

	private BLFacade createLocalBL() {
		DataAccess da = new DataAccess(c.getDataBaseOpenMode().equals("initialize"));
		return new BLFacadeImplementation(da);
	}

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

Como observamos, el metodo createBLFacade nos devuelve una implementacion de BLFacade, sin importar el tipo concreto. Es decir, se selecciona el tipo en tiempo de ejecucion y esta clase se encarga de crearlo y devolverlo al main en Application Launcher.

Diagrama UML - Solucion



2. Iterator

Código

Se crearon las clases:

- ***ExtendedIterator***: interfaz que define los métodos de un iterador extendido de objetos, y además extiende al *Iterator* de java.util con sus dos métodos *next() y hasNext():*

```
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();
}
```

- ***ExtendedIteratorEvents***: iterador concreto para eventos, que implementa la interfaz *ExtendedIterator* para este tipo de objetos:

```
public class ExtendedIteratorEvents implements ExtendedIterator<Event> {
	List<Event> events;
	int position = 0;

	public ExtendedIteratorEvents(List<Event> evs) {
		this.events = evs;
	}

	public boolean hasNext() {
		return position < events.size();
	}

	public Event next() {
		Event event = events.get(position);
		position = position + 1;
		return event;
	}

	public Event previous() {
		Event event = events.get(position);
		position = position - 1;
		return event;
```

```
	}

	public boolean hasPrevious() {
		return position >= 0;
	}

	public void goFirst() {
		position = 0;

	}

	public void goLast() {
		position = events.size() - 1;

	}

}
```

- ***IteratorMainProgram****:* una clase main (método main) que creamos únicamente para visualizar el funcionamiento del iterador al iterar tanto en orden creciente como decreciente:

```
public class IteratorMainProgram {

	public static void main(String[] args) {
		BLFacade blFacade = new
BLFactory(ConfigXML.getInstance()).createBLFacade(BL_IMPLEMENTATIONS.LOCAL);
		SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

		Date date;
		try {
			date = sdf.parse("18/12/2022");
			blFacade.gertaerakSortu("aDescription1-aDescription1", date,
"Futbol");
			blFacade.gertaerakSortu("aDescription2-aDescription2", date,
"Futbol");
			blFacade.gertaerakSortu("aDescription3-aDescription3", date,
"Futbol");
			blFacade.gertaerakSortu("aDescription4-aDescription4", date,
"Futbol");

			ExtendedIterator<Event> eventIterator =
blFacade.getEventsIterator(date);

			System.out.println("HACIA ATRAS:");
			eventIterator.goLast();
			Event actual;
			while (eventIterator.hasPrevious()) {
				actual = eventIterator.previous();
				System.out.println(actual.toString());
```

```
                        }

                        System.out.println("HACIA ADELANTE");
                        eventIterator.goFirst();
                        while (eventIterator.hasNext()) {
                                actual = eventIterator.next();
                                System.out.println(actual.toString());
                        }
                } catch (ParseException e1) {
                        System.out.println("Problem has occured: " + e1.getMessage());
                } catch (EventFinished e2) {
                        System.out.println("Problem has occured: " + e2.getMessage());
                }
        }
}
```

Además, se agregó el siguiente método en la interfaz ***BLFacade***:

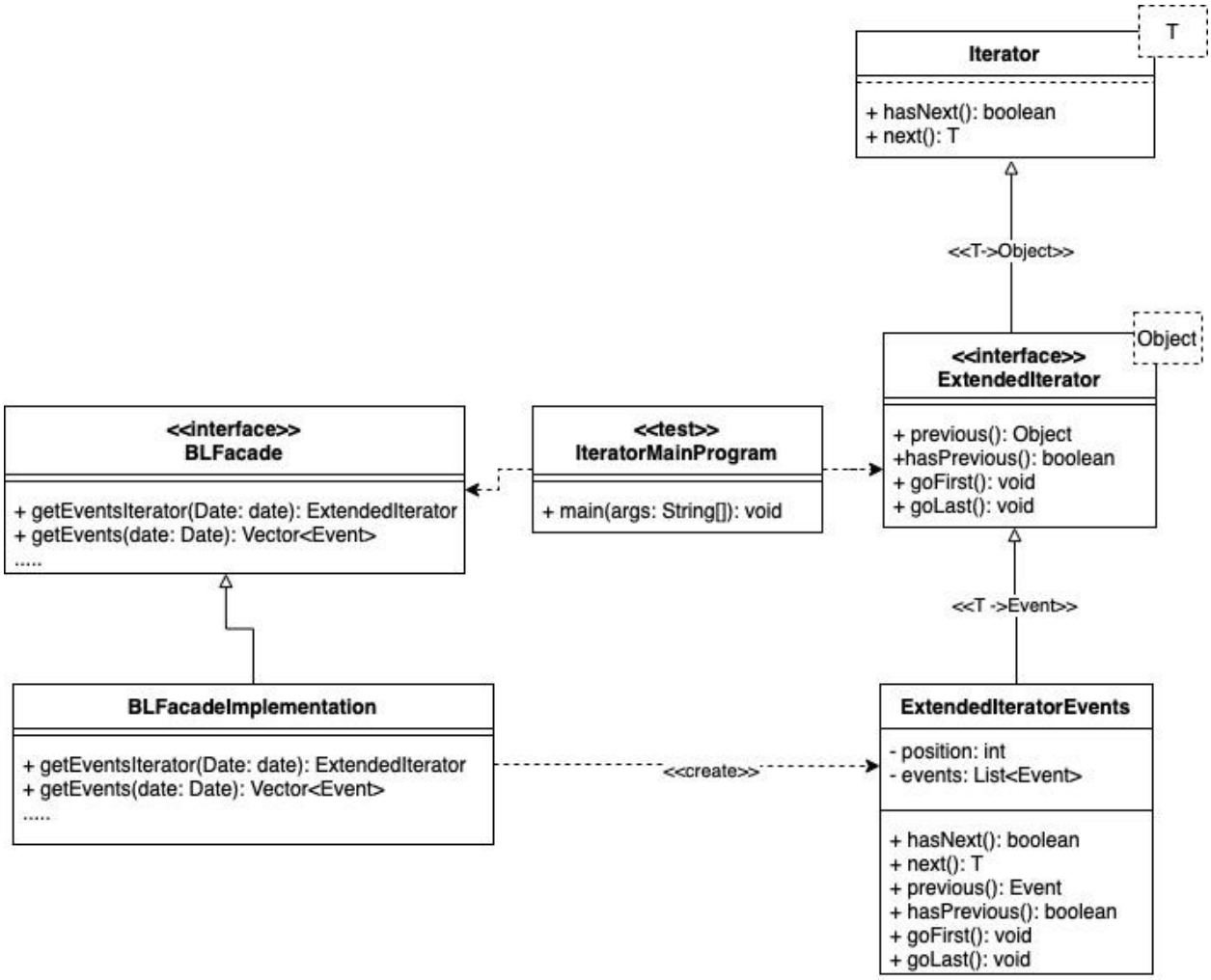
```
/**
         * This method retrieves the events of a given date
         *
         * @param date in which events are retrieved
         * @return extended iterator of events
         */
        @WebMethod public ExtendedIterator<Event> getEventsIterator(Date date);
```

Y este fue implementado en ***BLFacadeImplementation*** de la siguiente forma:

```
/**
         * This method retrieves the events of a given date
         *
         * @param date in which events are retrieved
         * @return extended iterator of events
         */
    public ExtendedIterator<Event> getEventsIterator(Date date) {
                return new ExtendedIteratorEvents(getEvents(date));
    }
```

Como vemos, para la implementación del método *getEventsIterator* reutilizamos el método que ya teníamos implementado *getEvents* que nos devuelve los eventos de determinada fecha. De esta forma, podemos crear el iterador extendido con esta respuesta, reutilizando código y cumpliendo con los lineamientos de clean code.

Diagrama UML



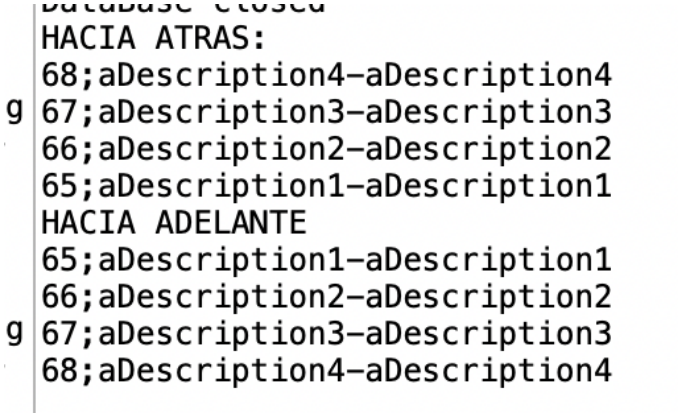
Aquí podemos ver la solución completa. Podemos ver que la interfaz *ExtendedIterator* es implementada por *ExtendedIteratorEvents* para construir el iterador concreto de eventos. En el futuro, se pueden implementar otros iteradores para otros tipos de objetos, sin impacto de cambio porque lo eunuco que tenemos que hacer es agregar una nueva clase y hacerla implementar la interfaz *ExtendedIterator,* cumpliendo así con OCP

A la clase BLFacade se le sumó el método getEventsIterator que nos devuelve el iterador, y este fue implementado en BLFacadeImplementation, que cumple el rol de **aggregate** de Event. Es por ello que la aquí se sitúa el método para la creación del iterador.

Luego tenemos el cliente *IteratorMainProgram* que consume el iterador y muestra por consola la correcta ejecución de este, que podemos ver a continuación:

Ejecución

La ejecución fue exitosa. Si vemos las capturas debajo, podemos observar el orden tanto creciente como decreciente en la iteración de los eventos. Agregamos 4 eventos numerados aDescription1, aDescription2, aDescription3, aDescription4 para ver esto más notoriamente:



3. Adapter

Se creó la clase RegisteredAdapter en el paquete de Domain que extiende de AbtractTableModel, para adaptar los métodos que utiliza para crear una tabla a nuestro caso en particular. De esta manera podemos crear la tabla de apuestas para un usuario sin tocar la clase de registered.

Código

src/main/java/domain/RegisteredAdapter.java

```
package domain;

import java.util.ArrayList;
import java.util.List;

import javax.swing.table.AbstractTableModel;

public class RegisteredAdapter extends AbstractTableModel {

private Registered registered;
private  List<Apustua> apustuak= new ArrayList();

public RegisteredAdapter(Registered r) {
       this.registered = r;
}

public int getRowCount() {
       for(ApustuAnitza aa: registered.getApustuAnitzak()) {
        for(Apustua a: aa.getApustuak()) {
                apustuak.add(a);
        }
}
       return apustuak.size();
}

public int getColumnCount() {
       return 4;
}

public Object getValueAt(int rowIndex, int columnIndex) {
       for(Apustua a: apustuak) {
              if(columnIndex==0) {
            return a.getKuota().getQuestion().getEvent();
              }
              else if(columnIndex==1) {
            return a.getKuota().getQuestion();
```

```
                }
                else if(columnIndex==2) {
            return a.getApustuAnitza().getData();
                }
                else {
            return a.getApustuAnitza().getBalioa();
                }
        }


        return null;
}

}
```

Aquí reescribimos los métodos que se llaman para crear una table model, indicamos el tamaño que queremos que tenga y que cargar en cada columna, recorriendo las apuestas del usuario y llenando cada columna para cada apuesta.

Diagrama UML - Solucion