Proyecto Patrones de diseño

Ingeniería de software II

Alejandro Sanchez Hernandez

Proyecto Github:

https://github.com/Delusion000/SI2_TEST.git

1) Factory

En la implementación del factory se determina que el creador es la clase factory la cual contiene unos métodos para este fin, tiene un método llamado createBLFacade que crea un objeto de la clase DBLFacade y otro método llamado createBusinessLogicFactory que decide si la implementación es local o remota.

El produc es la interface BLFacade que se instancia permitiendo el uso de los dos tipos de conexión de manera uniforme dependiendo de lo que solicite el usuario y por ultimo el concreteProduct son las clases que implementan BLFacade con un método createBLFacade que devuelve una instancia de BLFacade ya sea RemoteBLFacade o LocalBLFacade.

Diagrama UML

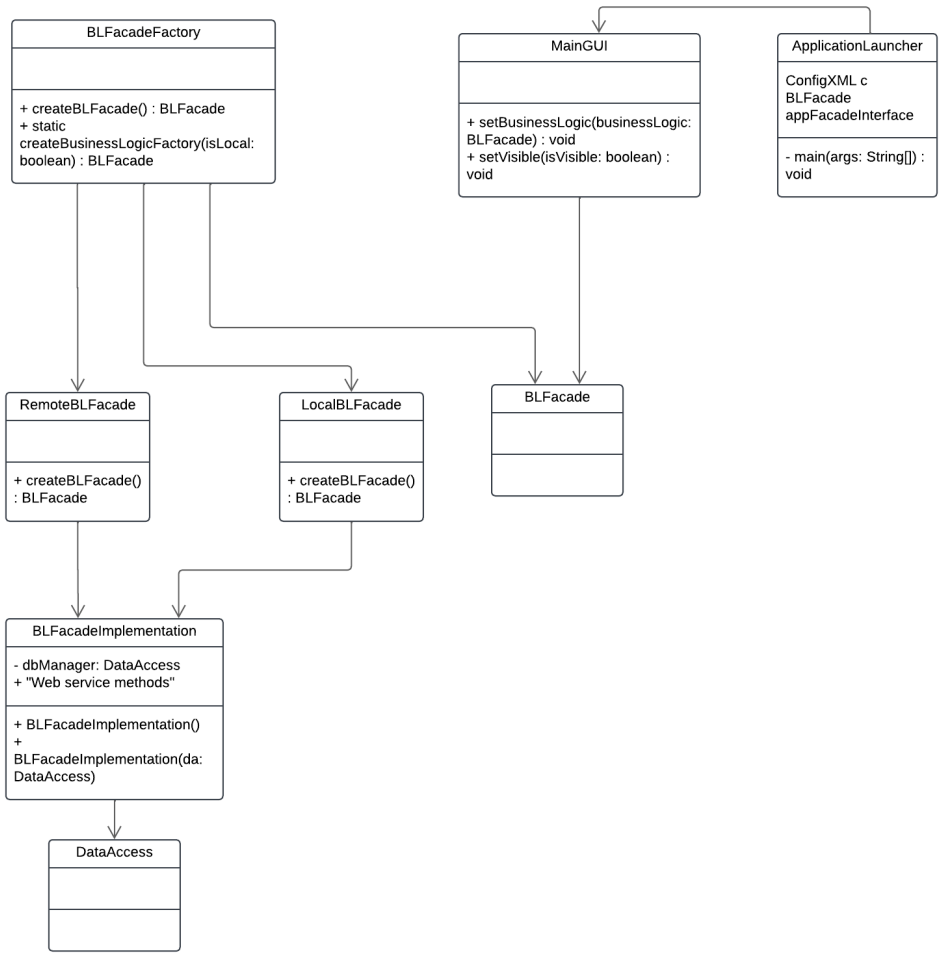


Imagen de la clase factory

```java
package businesslogic;

// Interfaz para crear una instancia de BLFacade
public interface BLFacadeFactory {
    BLFacade createBLFacade();

    // Método estático para crear BLFacade en función de si es local o remoto
    static BLFacade createBusinessLogicFactory(boolean isLocal) {
        BLFacadeFactory factory;

        if (isLocal) {
            factory = new LocalBLFacade();
        } else {
            factory = new RemoteBLFacade();
        }

        return factory.createBLFacade();
    }
}
```

Implementación en AplicationLauncher

```java
package gui;

import java.util.Locale;
import javax.swing.UIManager;
import businesslogic.BLFacade;
import businesslogic.BLFacadeFactory;
import configuration.ConfigXML;

public class ApplicationLauncher {
    public static void main(String[] args) {
        ConfigXML c = ConfigXML.getInstance();
        Locale.setDefault(new Locale(c.getLocale()));

        try {
            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

            BLFacade appFacadeInterface = BLFacadeFactory.createBusinessLogicFactory(c.isBusinessLogicLocal());

            MainGUI.setBussinessLogic(appFacadeInterface);
            MainGUI a = new MainGUI();
            a.setVisible(true);
        } catch (Exception e) {
            System.out.println("Error in ApplicationLauncher: " + e.toString());
        }
    }
}
```

Clases loca y remote

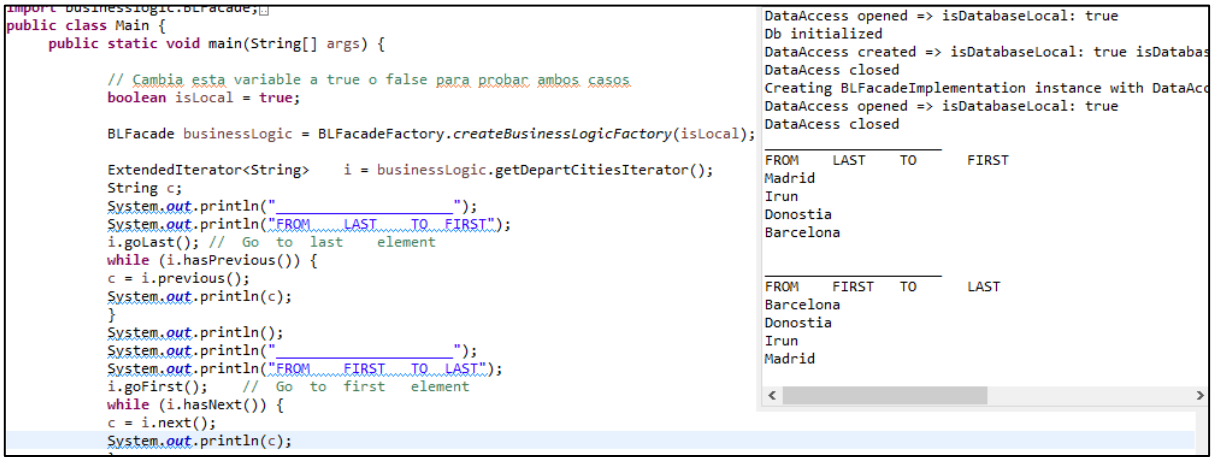
```java
package businesslogic;

import java.net.URL;

public class RemoteBLFacade implements BLFacadeFactory {
    @Override
    public BLFacade createBLFacade() {
        try {
            ConfigXML c = ConfigXML.getInstance();
            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);
            return service.getPort(BLFacade.class);
        } catch (Exception e) {
            System.out.println("Error creating remote BLFacade: " + e.toString());
            return null;
        }
    }
}
```

```java
package businesslogic;

import dataAccess.DataAccess;

public class LocalBLFacade implements BLFacadeFactory {
    @Override
    public BLFacade createBLFacade() {
        DataAccess da = new DataAccess();
        return new BLFacadeImplementation(da);
    }
}
```

2) Iterator

Para realizar el iterador extendido se crearon dos clases ExtendedIterator y AdapterExtendedIterator que convierte ListIterator en un ExtendedIterator facilitando la manipulación de los datos, también se modifico la clase BLFacadeImplementation agregándole un método getDepartCitiesIterator() crea y devuelve un ExtendedIterator para la lista de ciudades, de modo que el cliente pueda navegar la lista en ambas direcciones.

Imagen ejecución main iterator



```
import businessLogic.BLFacade;
public class Main {
    public static void main(String[] args) {

        // Cambia esta variable a true o false para probar ambos casos
        boolean isLocal = true;

        BLFacade businessLogic = BLFacadeFactory.createBusinessLogicFactory(isLocal);

        ExtendedIterator<String>    i = businessLogic.getDepartCitiesIterator();
        String c;
        System.out.println("_____________________");
        System.out.println("FROM    LAST    TO    FIRST");
        i.goLast(); //  Go  to  last    element
        while (i.hasPrevious()) {
        c = i.previous();
        System.out.println(c);
        }
        System.out.println();
        System.out.println("_____________________");
        System.out.println("FROM    FIRST    TO    LAST");
        i.goFirst();    //  Go  to  first   element
        while (i.hasNext()) {
        c = i.next();
        System.out.println(c);
```

```
DataAccess opened => isDatabaseLocal: true
Db initialized
DataAccess created => isDatabaseLocal: true isDatabas
DataAcess closed
Creating BLFacadeImplementation instance with DataAcc
DataAccess opened => isDatabaseLocal: true
DataAcess closed
_____________________
FROM    LAST    TO      FIRST
Madrid
Irun
Donostia
Barcelona

_____________________
FROM    FIRST    TO      LAST
Barcelona
Donostia
Irun
Madrid
```

Método añadido a la clase BLFacadeImplementation

```
public ExtendedIterator<String> getDepartCitiesIterator(){
    dbManager.open();

    List<String> departLocations = dbManager.getDepartCities();

    dbManager.close();

    return new AdapterExtendedIterator<>(departLocations);
}
```

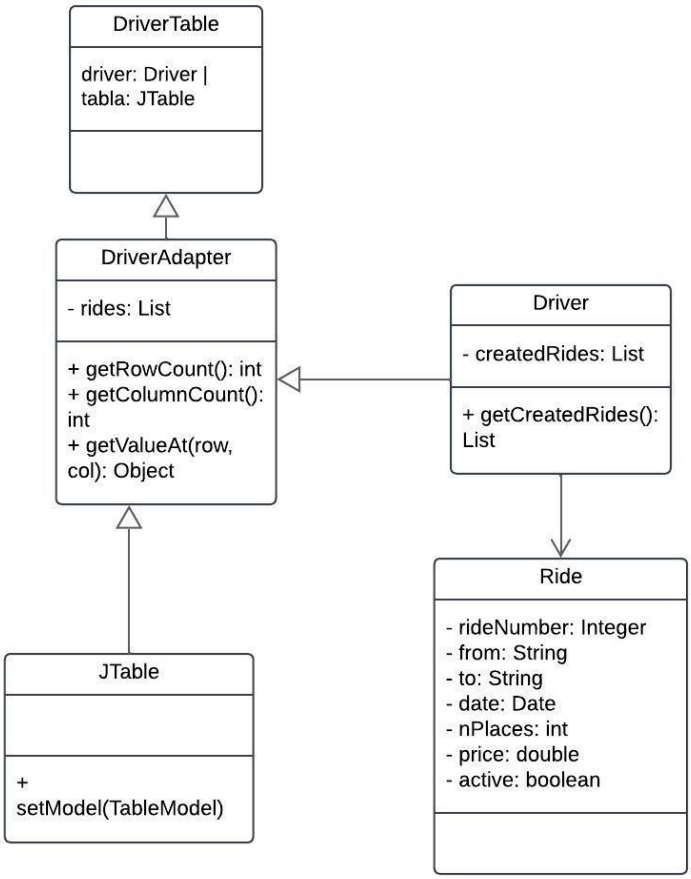
Clase ExtendedIterator

```
public class AdapterExtendedIterator<T> implements ExtendedIterator<T> {
    private ListIterator<T> iterator;
    private List<T> list;

    public AdapterExtendedIterator(List<T> list) {
        this.list = list;
        this.iterator = list.listIterator();
    }
```

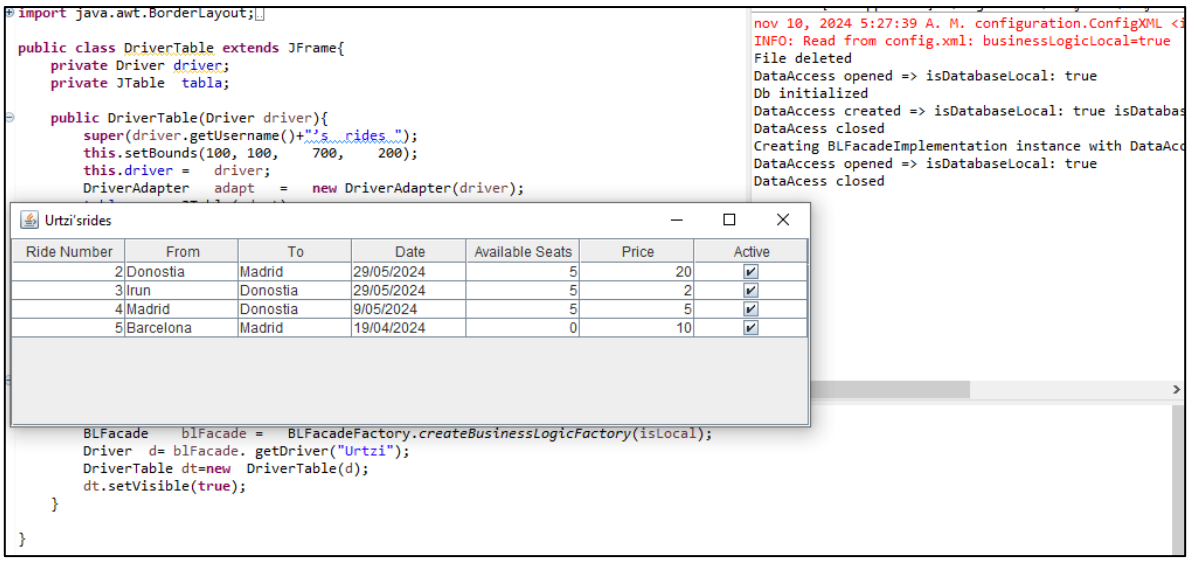
3) Adapter

En este patron para se crea una clase DriverAdapter la cual tiene la tarea de adaptar los datos de las clases RIDE y DRIVER y hacer uso de sus datos en la clase DriverTable, la clase adapter se encarga de conseguir una lista de rides de un objeto driver haciendo uso del método .getCreatedRides, luego define unas columnas las cuales se les aplica los diferentes datos usando métodos get de la clase ride para posteriormente presentarlos en la tabla DriverTable por medio de una instancia de DriverAdapter.

Diagrama UML



Ejecución y tabla



Clase DriverAdapter

```
    private final List<Ride> rides;

    // Definimos los nombres de las columnas
    private final String[] columnNames = { "Ride Number", "From", "To", "Date", "Available Seats", "Price", "Active" };

    public DriverAdapter(Driver driver) {
        this.rides = driver.getCreatedRides();
    }

    @Override
    public int getRowCount() {
        return rides.size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Ride ride = rides.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return ride.getRideNumber().shortValue();
            case 1:
                return ride.getFrom();
            case 2:
                return ride.getTo();
            case 3:
                return ride.getDate();
            case 4:
                return ride.getnPlaces();
            case 5:
                return ride.getPrice();
            case 6:
                return ride.isActive();
```

Clase DriverTable

```
public class DriverTable extends JFrame{
    private Driver driver;
    private JTable  tabla;

    public DriverTable(Driver driver){
        super(driver.getUsername()+"'s  rides ");
        this.setBounds(100, 100,    700,    200);
        this.driver =   driver;
        DriverAdapter   adapt   =   new DriverAdapter(driver);
        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);
    }


    public static void main(String[]    args)   {
        //  the BL  is  local
        boolean isLocal =   true;
        BLFacade    blFacade =   BLFacadeFactory.createBusinessLogicFactory(isLocal);
        Driver  d= blFacade. getDriver("Urtzi");
        DriverTable dt=new  DriverTable(d);
        dt.setVisible(true);
    }
```