Patrones de diseño:

Rides

Autores:

- Rubén Gallego
- Mikel Berasategui

Asignatura: Ingeniería de Software II

Enlaces del proyecto:

-Github: https://github.com/RubenGGBC/rides

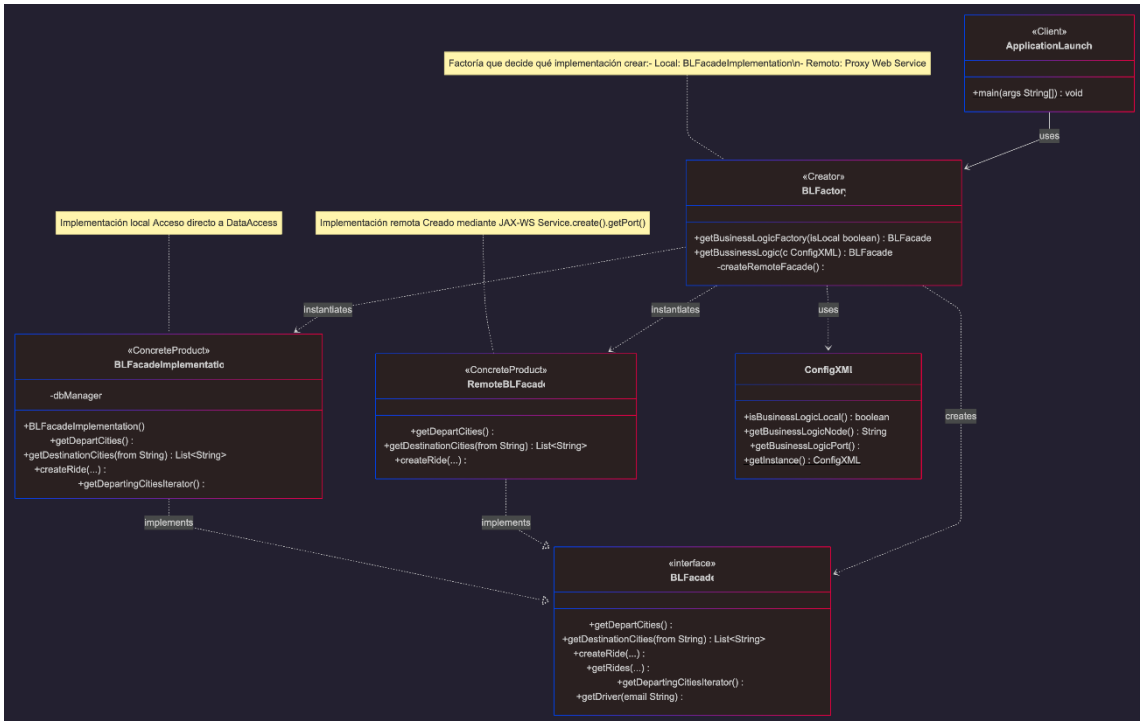
Índice

1.Patrón Factory Method

Se pide:

Modifica la aplicación para que la obtención del objeto de la lógica de negocio se centralice en un objeto factoría, y sean las clases de la presentación las que decidan cuál de las implementaciones de la lógica de negocio utilizar. Diseña e implementa la solución indicando qué clases juegan el papel de Creator, Product y ConcreteProduct.

1.1-Diagrama UML



1.2-Código Modificado

- *BLFactory:* Es la clase encargada de centralizar la creación de objetos BLFacade, aquí se decide si hacer una instancia remota o local.

```java
package gui;

import businessLogic.BLFacade;
import businessLogic.BLFacadeImplementation;
import configuration.ConfigXML;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.MalformedURLException;
import java.net.URL;

public class BLFactory {  5 usages  new *

    public BLFacade getBusinessLogicFactory(boolean isLocal) {  3 usages  new *
        if (isLocal) {
            return new BLFacadeImplementation();
        } else {
            return createRemoteFacade();
        }
    }

    public BLFacade getBussinessLogic(ConfigXML c) { return getBusinessLogicFactory(c.isBusinessLogicLocal()); }

    private BLFacade createRemoteFacade() {  1 usage  new *
        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( namespaceURI: "http://businessLogic/",  localPart: "BLFacadeImplementationService");
            Service service = Service.create(url, qname);

            return service.getPort(BLFacade.class);
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }

}
```

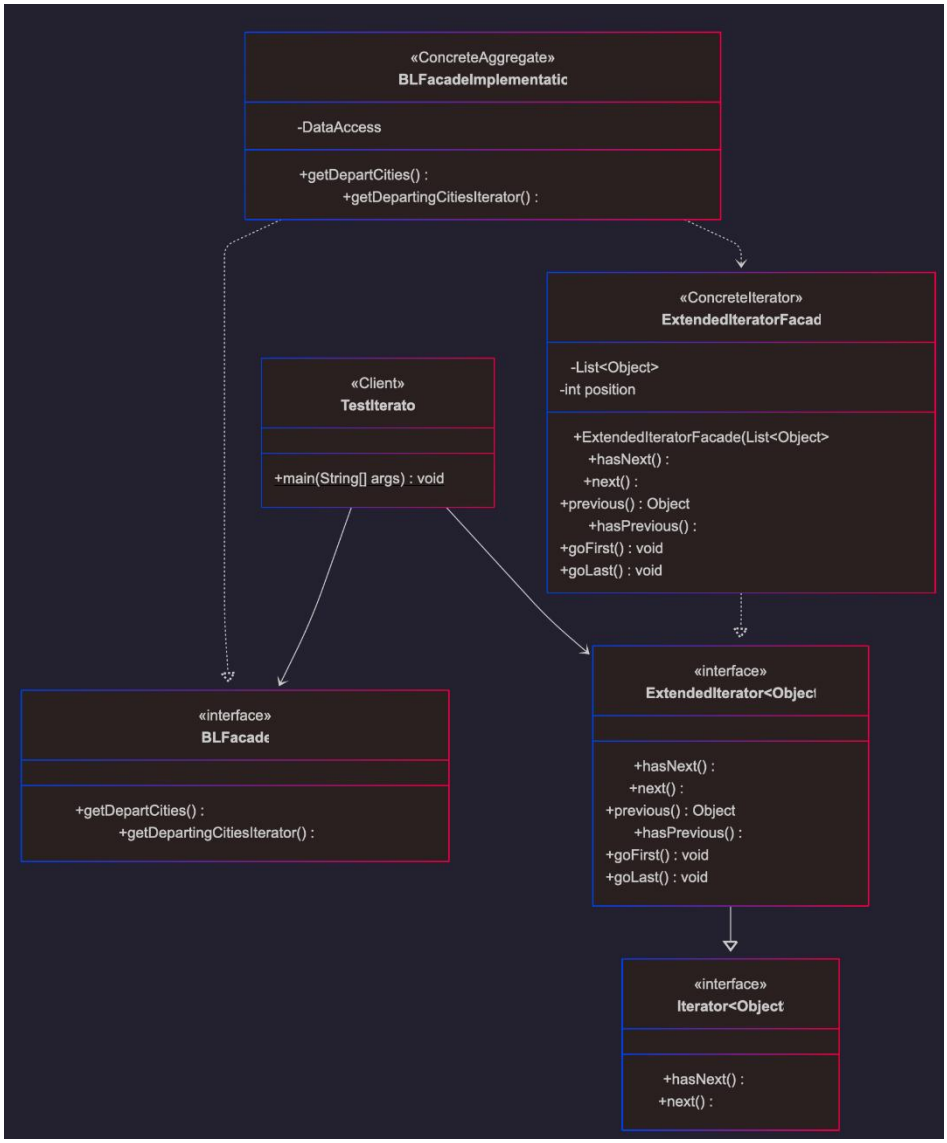
- *ApplicationLauncher:* Clase principal que ahora delega la creación de BLFacade a la factoría, simplificando su responsabilidad.

2. Patrón Iterator

Se pide:

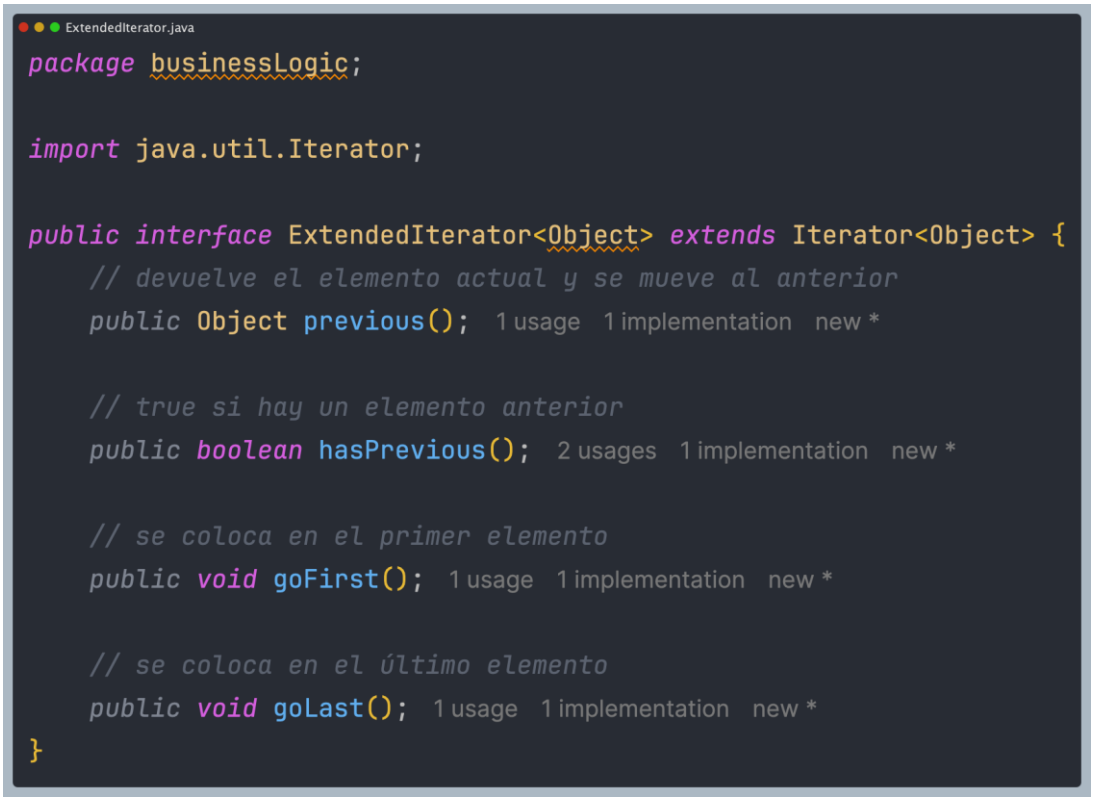
Implementar el Iterator extendido, y realizar un programa similar al del ejemplo que visualice todos los eventos en orden creciente y decreciente.

2.1-Diagrama UML



2.2-Código Modificado

- *ExtendedIterator:* Interfaz que extiende Iterator que permite moverse en ambos sentidos mediante los métodos previous(), hasPrevious(), goFirst() y goLast().



- *ExtendedIteratorFacade*: Implementación del iterador que mantiene una lista y una posición actual, permitiendo recorrer la colección en ambas direcciones:

ExtendedIteratorFacade.java

```java
package businessLogic;

import ...

public class ExtendedIteratorFacade implements ExtendedIterator<Object> {
    private List<Object> list;  5 usages
    private int position;  9 usages

    public ExtendedIteratorFacade(List<Object> list) {  13 usages  new *
        this.list = list;
        this.position = -1;
    }

    @Override  1 usage  new *
    public Object previous() {
        if (!hasPrevious()) {
            throw new NoSuchElementException();
        }
        position--;
        return list.get(position);
    }

    @Override  2 usages  new *
    public boolean hasPrevious() { return position > 0; }

    @Override  1 usage  new *
    public void goFirst() { position = -1; }

    @Override  1 usage  new *
    public void goLast() { position = list.size(); }

    @Override  new *
    public boolean hasNext() { return position < list.size() - 1; }

    @Override  new *
    public Object next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        position++;
        return list.get(position);
    }
}
```

- *BLFacade*: En esta clase simplemente añadimos el nuevo método que implementaremos en el facade:

```java
BLFacade.java
    public ExtendedIterator<String> getDepartingCitiesIterator();
}
```

- *BLFacadeImplementation*: Implementamos el método que nos devolverá el nuevo Iterador:

```java
BLFacadeImplementation.java
public ExtendedIterator<String> getDepartingCitiesIterator(){  1 usage  new *
    dbManager.open();
    List<String> cities = dbManager.getDepartCities();
    dbManager.close();

    List<Object> objectList = new ArrayList<>();
    for (String city : cities) {
        objectList.add(city);
    }

    return (ExtendedIterator<String>)(ExtendedIterator<?>) new ExtendedIteratorFacade(objectList);
}
```

2.3-Ejecución del resultado

- *Main* de pruebas (La clase esta en la carpeta de test):

TestIterator.java

```java
import ...

public class TestIterator {
    public static void main(String[] args) {
        boolean isLocal = true;
        BLFacade blFacade = new BLFactory().getBusinessLogicFactory(isLocal);
        ExtendedIterator<String> i = blFacade.getDepartingCitiesIterator();
        String c;
        System.out.println("____________________");
        System.out.println("DEL ULTIMO AL PRIMERO");
        i.goLast();
        while (i.hasPrevious()) {
            c = i.previous();
            System.out.println(c);
        }
        System.out.println();
        System.out.println("____________________");
        System.out.println("DEL PRIMERO AL ULTIMO");
        i.goFirst();
        while (i.hasNext()) {
            c = i.next();
            System.out.println(c);
        }
    }
}
```

- Ejecución:

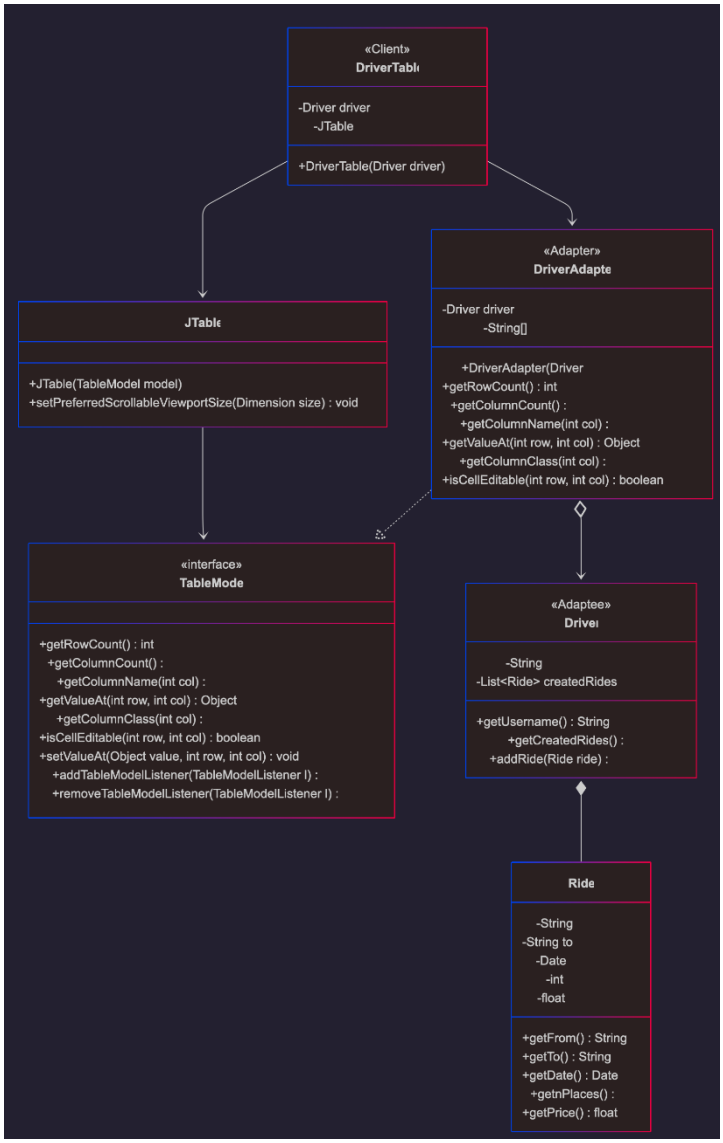
```
INFO: Creating BLFacadeImplementation instance
Read from config.xml:    businessLogicLocal=true     databaseLocal=true  dataBaseInitialized=false
DataAccess opened => isDatabaseLocal: true
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: false
DataAcess closed
DataAccess opened => isDatabaseLocal: true
DataAcess closed
--------------------
DEL ULTIMO AL PRIMERO
Lumiere
Donostia

--------------------
DEL PRIMERO AL ULTIMO
Donostia
Lumiere
```

3. Patrón Adapter

Se pide:

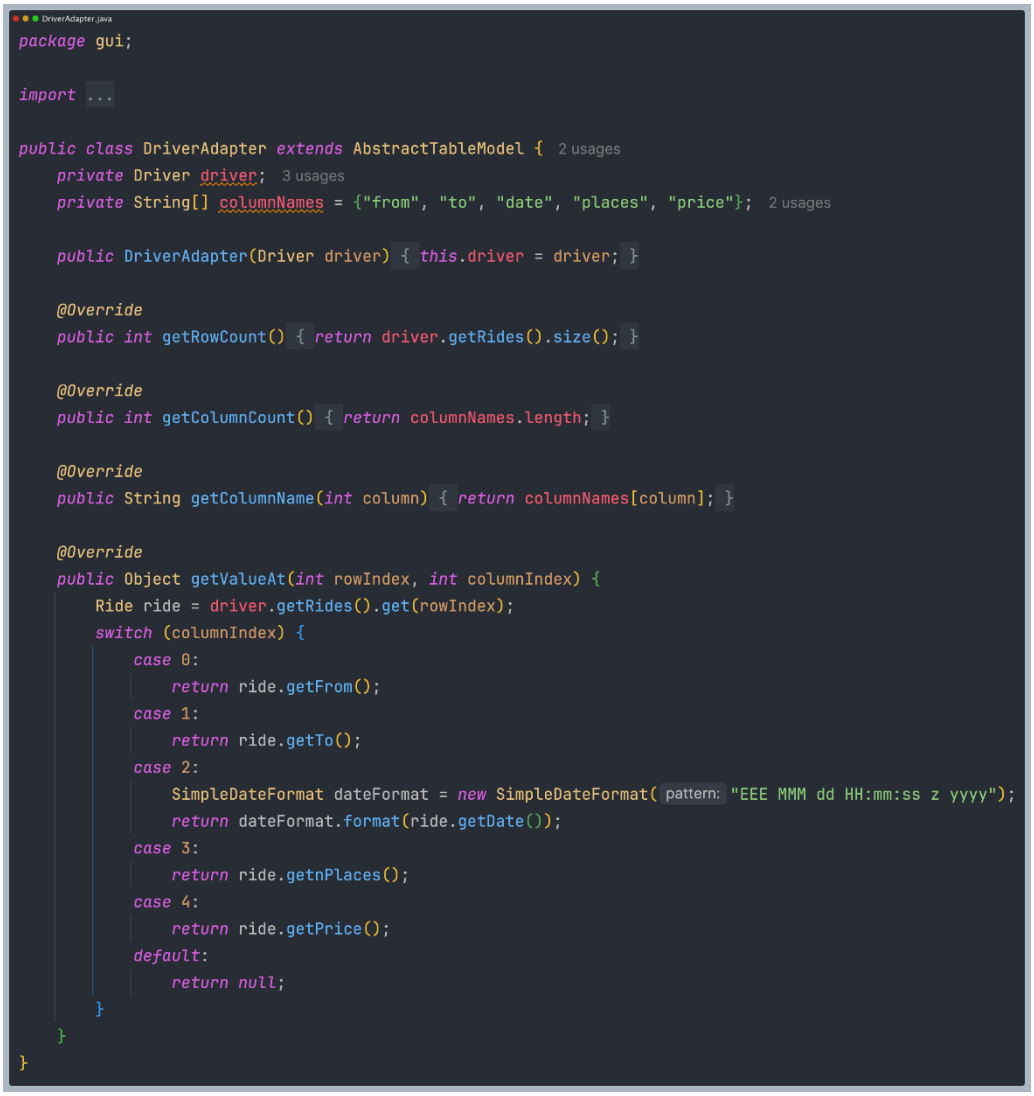
Implementar una clase adaptadora DriverAdapter de tipo TableModel, para poder visualizar un objeto de Driver (sus rides) en una JTable (Tambien hay que entregar el diagrama UML indicando qué papel juega cada clase en el patrón adapter).

3.1-Diagrama UML



3.2-Código Modificado

- *DriverAdapter*: Adapta un objeto Driver para que sea compatible con JTable. Implementa los métodos de AbstractTableModel para mostrar los viajes en formato tabla.



```java
package gui;

import ...

public class DriverAdapter extends AbstractTableModel {  2 usages
    private Driver driver;  3 usages
    private String[] columnNames = {"from", "to", "date", "places", "price"};  2 usages

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

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

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

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

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Ride ride = driver.getRides().get(rowIndex);
        switch (columnIndex) {
            case 0:
                return ride.getFrom();
            case 1:
                return ride.getTo();
            case 2:
                SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "EEE MMM dd HH:mm:ss z yyyy");
                return dateFormat.format(ride.getDate());
            case 3:
                return ride.getnPlaces();
            case 4:
                return ride.getPrice();
            default:
                return null;
        }
    }
}
```

- *DriverTable*: JFrame que usa DriverAdapter para visualizar los viajes de un conductor en una JTable con scroll.

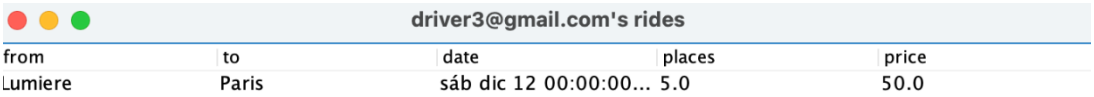
```java
package gui;

import ...

public class DriverTable extends JFrame {  3 usages
    private Driver driver;  1 usage
    private JTable tabla;  3 usages

    public DriverTable(Driver driver) {  14 usages
        super( title: driver.getEmail() + "'s rides");
        this.setBounds( x: 100,  y: 100,  width: 700,  height: 200);
        this.driver = driver;
        DriverAdapter adapt = new DriverAdapter(driver);
        tabla = new JTable(adapt);
        tabla.setPreferredScrollableViewportSize(new Dimension( width: 500,  height: 70));
        JScrollPane scrollPane = new JScrollPane(tabla);
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }
}
```

3.3-Ejecución del resultado

- *Main* de pruebas (La clase este en la carpeta de test):

```java
import ...

public class TestAdapter {
    public static void main(String[] args) throws UserAlredyExistException, RideAlreadyExistException, RideMustBeLaterThanTodayException {
        boolean isLocal = true;
        BLFacade blFacade = new BLFactory().getBusinessLogicFactory(isLocal);

        Date testDate = new Date( year: 126,  month: 11,  date: 12);
        Ride createdRide = blFacade.createRide( from: "Lumiere",  to: "Paris", testDate,  nPlaces: 5,  price: 50,  driverEmail: "driver3@gmail.com");

        Driver d = blFacade.getDriver( email: "driver3@gmail.com");
        DriverTable dt = new DriverTable(d);

        dt.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                d.removeRide( from: "Lumiere",  to: "Paris", testDate);
                blFacade.updatearDriver(d);
                System.out.println("Viaje de prueba eliminado de la BD");
                System.exit( status: 0);
            }
        });

        dt.setVisible(true);
    }
}
```

- Ejecución:


driver3@gmail.com's rides
from
to
date
places
price
Lumiere
Paris
sáb dic 12 00:00:00...
5.0
50.0