

# PROYECTO PATRONES

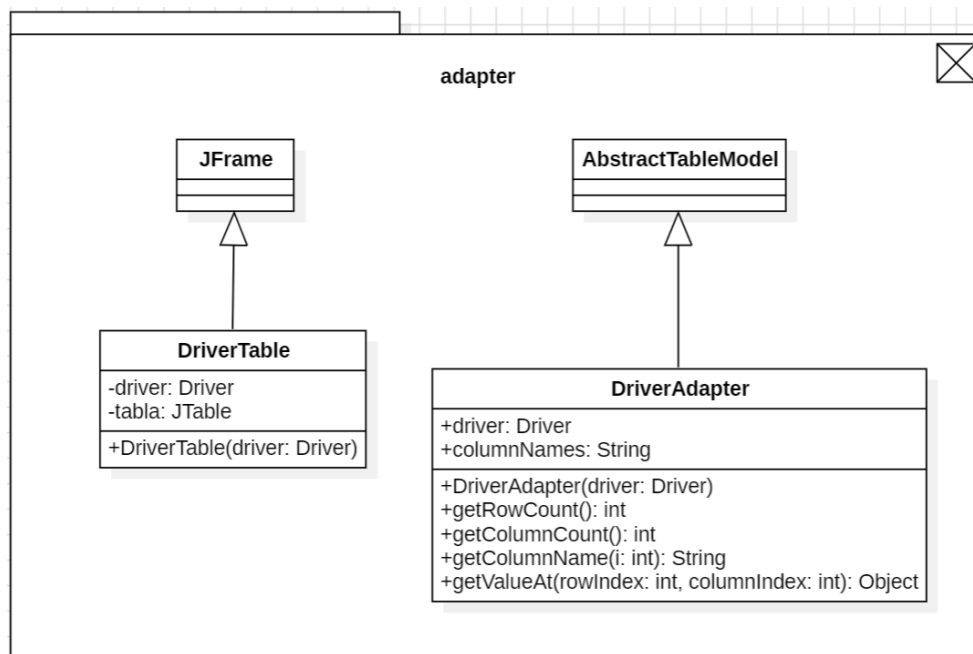
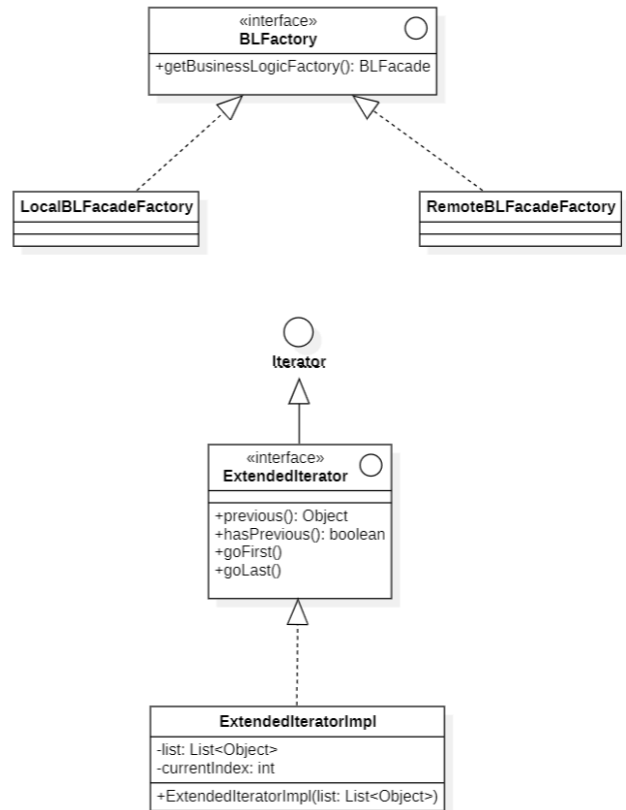
Paula Recaj, Irati Cruz y Amna Rumaisa

URL: <https://github.com/PaulaRecaj/ridesIS2>

A) Un diagrama UML extendido, indicando los cambios realizados y la finalidad de cada clase/interfaz.

- Interfaz **BLFactory** (creator): contiene el método `getBusinessLogicFactory()`.
- Clase **LocalBLFacadeFactory** (concrete creator): clase que implementa la interfaz `BLFactory`. Tiene como finalidad realizar las acciones necesarias en el método `getBusinessLogicFactory()` cuando ha sido comprobado que `isBusinessLogicLocal` es true.
- Clase **RemoteBLFacadeFactory** (concrete creator): clase que implementa la interfaz `BLFactory`. Tiene como finalidad realizar las acciones necesarias en el método `getBusinessLogicFactory()` cuando ha sido comprobado que `isBusinessLogicLocal` es false.
- Clase **ExtendedIteratorImpl**: implementa la interfaz `ExtendedIterator` y desarrolla los métodos `next`, `hasNext`, `previous`, `hasPrevious`, `goFirst` y `goLast`.
- Clase **DriverAdapter**: para poder visualizar los viajes de un `Driver` en una `JTable` hemos creado la clase adaptadora `DriverAdapter` la cual se encarga de convertir los datos del `Driver` y sus `Rides` en un formato que `JTable` pueda entender y mostrar, se encarga de definir las columnas y los datos de cada fila que son los viajes creados por el `Driver`.
- Clase **DriverTable**: se encarga de crear la ventana gráfica `JFrame` para mostrar los datos proporcionados por `DriverAdapter`.
- Clase **Main**: Hemos creado un `main` que se encuentra en el package `adapter` que ejecuta la aplicación mostrando en un `JFrame` los viajes del `Driver`.

# businessLogic



B) El código que habéis modificado, describiendo las líneas más significativas.

### 1. Patrón Factory Method

- En vez de realizar todo en el ApplicationLauncher, se crea la interfaz BLFactory y las clases LocalBLFacadeFactory y RemoteBLFacadeFactory para así aplicar el método Factory.

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

        ConfigXML c = ConfigXML.getInstance();

        Logger l = Logger.getLogger(ConfigXML.class.getName());
        l.config(c.getLocale());
        Locale.setDefault(new Locale(c.getLocale()));
        l.config("Locale: " + Locale.getDefault());

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

            if (c.isBusinessLogicLocal()) {

                factory = new LocalBLFacadeFactory();

            } else {

                factory = new RemoteBLFacadeFactory();

            }

            BLFacade appFacadeInterface = factory.getBusinessLogicFactory();
            MainGUI.setBusinessLogic(appFacadeInterface);
            MainGUI a = new MainGUI();
            a.setVisible(true);

        } catch (Exception e) {
            l.config("Error in ApplicationLauncher: " + e.toString());
        }
    }
}

public interface BLFactory {
    BLFacade getBusinessLogicFactory ();
}
```

```
public class LocalBLFacadeFactory implements BLFactory{
    @Override
    public BLFacade getBusinessLogicFactory() {
        DataAccess da = new DataAccess();
        return new BLFacadeImplementation(da);
    }
}
```

```
public class RemoteBLFacadeFactory implements BLFactory{
    @Override
    public BLFacade getBusinessLogicFactory() {
```

```

        try {
            ConfigXML c = ConfigXML.getInstance();
            String serviceName = "http://" + c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort()
                + "/ws/"
                + c.getBusinessLogicName() + "?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/", "BLFacadeImplementationService");
            Service service = Service.create(url, qname);
            return service.getPort(BLFacade.class);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

## 2. Patrón Iterator

- Se crea la clase **ExtendedIteratorImpl** para implementar la interfaz **ExtendedIterator**. En esta, se desarrollan los métodos necesarios para iterar sobre las ciudades.

```

public class ExtendedIteratorImpl<Object> implements ExtendedIterator<Object> {
    private List<Object> list;
    private int currentIndex;
    public ExtendedIteratorImpl(List<Object> list) {
        this.list = list;
        this.currentIndex = 0;
    }

    @Override
    public boolean hasNext() {
        return currentIndex < list.size();
    }

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

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

    @Override
    public boolean hasPrevious() {
        return currentIndex >= 0;
    }
}

```

```

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

@Override
public void goLast() {
    currentIndex = list.size() - 1;
}

```

```

/**
 * {@inheritDoc}
 */
@WebMethod
public ExtendedIterator<String> getDepartCitiesIterator() {
    dbManager.open();
    List<String> departLocations = dbManager.getDepartCities();
    dbManager.close();
    ExtendedIterator<String> res = new ExtendedIteratorImpl(departLocations);
    return res;
}

```

### 3. Patrón Adapter

Para poder mostrar los viajes de un driver en un JTable hemos creado un package llamado **adapter** donde se encuentran la clase **Main**, **DriverTable** y **DriverAdapter**.

```

package adapter;

import javax.swing.table.AbstractTableModel;

import domain.Driver;
import domain.Ride;

public class DriverAdapter extends AbstractTableModel {

    private Driver driver;
    private String[] columnNames = new String[] { "from", "to", "date", "places", "price" };

    public DriverAdapter(Driver driver) {

        this.driver = driver;
    }

    @Override
    public int getRowCount() {
        // TODO Auto-generated method stub
        return driver.getCreatedRides().size();
    }

    @Override
    public int getColumnCount() {
        // TODO Auto-generated method stub
    }
}

```

```

        return columnNames.length;
    }

    public String getColumnName(int i) {
        // Challenge!
        return columnNames[i];
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {

        Ride r = (Ride)driver.getCreatedRides().toArray()[rowIndex];

        switch(columnIndex) {

            case 0:
                return r.getFrom();
            case 1:
                return r.getTo();
            case 2:
                return r.getDate();
            case 3:
                return r.getnPlaces();
            case 4:
                return r.getPrice();

            default:
                return null;

        }

    }

}

```

```

package adapter;

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

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

import domain.Driver;

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);
    }
}

```

```

package adapter;

import businesslogic.*;
import domain.Driver;

public class Main {

    public static void main(String[] args) {
        // the BL is local
        //boolean isLocal = true;
        //BLFacade blFacade = new BLFactory().getBusinessLogicFactory(isLocal);
        BLFacade blf = new BLFacadeImplementation();
        LocalBLFacadeFactory localblf = new LocalBLFacadeFactory();

        blf.initializeBD();

        Driver d= blf.getDriver("Urtzi");
        DriverTable dt=new DriverTable(d);
        dt.setVisible(true);
    }
}

```

C) Para los patrones Iterator y Adapter, una captura de imagen que muestre su ejecución.

## 2. Patrón Iterator

```
.DataAccess opened => isDatabaseLocal: true
.DataAccess created => isDatabaseLocal: true isDatabaseInitialized: false
.DataAccess closed
.DataAccess opened => isDatabaseLocal: true
.DataAccess closed

FROM LAST TO FIRST
Madrid
Irun
Donostia
Barcelona

FROM FIRST TO LAST
Barcelona
Donostia
Irun
Madrid
```

## 3. Patrón Adapter

Urtzi's rides				
from	to	date	places	price
Donostia	Madrid	Thu May 30 00:00:00 CEST 2024	5	20.0
Irun	Donostia	Thu May 30 00:00:00 CEST 2024	5	2.0
Madrid	Donostia	Fri May 10 00:00:00 CEST 2024	5	5.0
Barcelona	Madrid	Sat Apr 20 00:00:00 CEST 2024	0	10.0